LLAMA (Lincoln Laboratory Advanced MARTHA Applications) Software Manual

D.W. White

1 December 2003

Lincoln Laboratory

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LEXINGTON, MASSACHUSETTS

20031216 183

This report may be reproduced to satisfy needs of U.S. Government agencies.

The ESC Public Affairs Office has reviewed this report, and it is releasable to the National Technical Information Service, where it will be available to the general public, including foreign nationals.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER

Gary Tutungian
Administrative Contracting Officer
Plans and Programs Directorate
Contracted Support Management

Massachusetts Institute of Technology
Lincoln Laboratory

LLAMA (Lincoln Laboratory Advanced MARTHA Applications) Software Manual

D.W. White
Group 601

Technical Report 1088

1 December 2003

Lexington   Massachusetts

ABSTRACT

For the past 25 years or more, a number of staff members at MIT Lincoln Laboratory have made extensive use of the *APL* computer language to solve a variety of problems, primarily in the area of radio frequency and microwave circuit design. This was aided and inspired by the availability of the *MARTHA* software package, which is a collection of *APL*-based circuit analysis functions developed by Professor Paul Penfield Jr. at MIT.

The Lincoln Laboratory Advanced *MARTHA* Applications (or *LLAMA* for short) is a set of 15 workspaces (a collection of *APL* functions) developed primarily in conjunction with *MARTHA*. Many of the workspaces are an extension of *MARTHA*, and allow the use of new circuit elements or new types of analysis. A number of workspaces are devoted to filter synthesis, using both lumped elements and coupled microstrip transmission lines. Other workspaces are aimed toward RF system design, including mixer-spur and dynamic-range analysis.

This manual is intended to provide more formal documentation for this resource than has previously been available. It is hoped that it will allow new users to quickly make use of all that *APL* and *MARTHA* have to offer, as well as providing a concise, well-indexed reference for the more experienced user.

ACKNOWLEDGEMENTS

LLAMA is the result of over 2 decades of programming by a large number of individuals at MIT Lincoln Laboratory. To the extent that it could be traced or remembered, credit for individual functions is given in each section. Many of these functions were originated long before the author of this manual arrived at Lincoln Lab, and many of the original authors of the mainframe *APL* have either retired or left for other opportunities. When much of this code was rescued from the Laboratory's mainframe computer, it was quite a mix of programming styles and the level of documentation varied wildly, including information on who had done what. Every attempt has been made to give credit where it is due, and any omissions are solely due to lack of information. Any function not specifically mentioned is probably the author's own work.

Particular thanks for much of what is here go to two people. First and foremost is Prof. Paul Penfield, Jr. at MIT, whose *MARTHA* started it all. His continued work produced a number of essential *LLAMA* functions, and his support was invaluable in rescuing *MARTHA* from the mainframe and getting it working on personal computers. The second person is David Hodsdon, who re-introduced the author to *MARTHA* when he first arrived at Lincoln Laboratory, and who wrote a large number of the original functions on which *LLAMA* is based.

I would also like to thank Mark Fishman for having the patience to carefully proofread the final version of this manual. He efforts have made this a much more polished document than it might have been otherwise.

Table of Contents

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

1.1 HISTORY

For the past 25 years or more, a number of staff members at MIT Lincoln Laboratory have made extensive use of the *APL* computer language to solve a variety of problems, primarily in the area of RF and microwave circuit design. This was aided and inspired by the availability of the *MARTHA* software package [1.1, 1.2], which is a library of *APL*-based circuit analysis functions developed by Professor Paul Penfield Jr. at MIT.

1.2 WHAT IS *APL*?

In order to appreciate the full value of *MARTHA* and the workspaces described in this manual, a few words about the *APL* language itself are in order. *APL* was originated at IBM as a concise form of mathematical notation. It became quite popular within the technical ranks of IBM, and a desire to make it machine executable resulted in the core of the computer language used today. Drawing from conventional math notation, a special character set was used. This required special terminals and printers, which was a hindrance to its acceptance and growth. Despite this, it developed a devoted (but somewhat small) following in research laboratories around the world. It also became popular for statistical and actuarial work at a number of large corporations. It is still taught at a number of Universities, and is popular in Europe, Canada and Russia.

Up until the mid-1980's, most *APL* was run on large mainframe computers. With the advent of personal computers, and an *APL* interpreter for them, it became possible to use programmable fonts and dot-matrix printers to deal with the character set problem. With the advent of 32-bit architectures, it is now possible to run larger programs in less time on a basic PC than was ever possible on the best mainframes of yore.

APL has a number of key features that make it an efficient problem solving tool:

1) *APL* programs are written using 'functions'. A function is a modular routine that can be executed in isolation, or it can be called by other functions (including itself). This makes well-written code re-useable. In this regard, *APL* has a lot in common with the 'C' programming language. It also means that small functions that perform well-defined tasks become an extension to the language itself. A function to calculate the characteristics of coupled microstrip transmission-lines can later find itself called as an essential ingredient of several different microwave filter synthesis functions. The utility of a function is frequently limited not by the intent of its original author, but only by the ingenuity of later users. Of course, this only works if the functions are documented well enough that they can be understood by other users.

2) *APL* is an *array* based language. In most computer languages, vectors, matrices and higher order arrays must be handled on an element-by-element basis. In *APL*, arrays are manipulated as arrays. If two (equal-length) vectors, A & B, need to be added, they are simply added: $A + B$, without the need for loops, indices etc. *APL* includes a number of powerful single-character operators to perform more complex manipulations of matrices and larger arrays, including transposition, rotation and inversion.

1

3) *APL* is a 'smart' language. It is smart enough to know how to handle large variables, so you never have to dimension arrays or allocate and de-allocate memory. It is smart enough to know that 'A' is a character, 1.25 is a floating point number, 2 is an integer and (depending on its usage) 1 is a Boolean. All numerical computations are automatically done in double precision. This eliminates an enormous amount of programming overhead and debugging.

4) The code is interpreted rather than compiled. This is viewed as a major drawback by many, and <u>can</u> result in slightly slower execution[*]. However, it means that *APL* can be used as a 'super-calculator' for quick solutions of technical problems. Operating in 'immediate mode', code typed on the screen is executed on the spot. This is not only handy for small computations, but by testing the execution of more complicated code, the pieces of a function can be debugged as they are built up. Rather than putting an entire function together, compiling and debugging after the fact, critical operations are checked 'in process'. The completed function frequently requires no further debugging.

MARTHA <u>is</u> a powerful circuit analysis package, but many of its capabilities are now available in other commercial circuit analysis products. However, its *real* utility is as an enormous library of circuit analysis building blocks (functions). If all that is required is the S-parameters of a network, there are a number of software packages on the market that can do the job just as well, and that have more 'user friendly' input and output. On the other hand, if you want to analyze something that has never been done before or synthesize a new kind of circuit, *APL* (with *MARTHA*) can do <u>anything</u> you can imagine. *MARTHA* should not be thought of as a circuit analysis package, but as an extension to a powerful programming language that makes it especially suited to circuit design.

LLAMA is not only a collection of (hopefully) useful workspaces, but it is also an example of the power of *APL* and *MARTHA* applied to real-world problems. The workspaces cover a broad range of tasks in RF and microwave circuit design. Some make extensive use of *MARTHA* functions, some (like mixer spur analysis) don't use *MARTHA* at all.

1.3 *LLAMA* AND DIFFERENT VERSIONS OF *APL*

The initial commercial release of *LLAMA* is intended to be used with APL2000's *APLSE* (Special Edition), which is freeware. The workspaces will also run under the old Manugistic (STSC) *APL★PLUS/PC*, which is now sold by APL2000 as *APL+PC*. *APL+PC* comes with full

---

[*] The slower execution of interpreted code is frequently used to dismiss *APL* out of hand for solving large problems. Because it operates on arrays and performs operations like matrix inversion internally, well written *APL* can be surprisingly fast. A large amount of the overhead of a language like BASIC results from re-interpreting code in loops. Using the array-based structure of *APL*, loops can frequently be avoided altogether, resulting in fast execution. As an example, a microstrip analysis program was translated from FORTRAN to *APL*, with all of the loops left intact. This took almost a full minute to analyze a single pair of coupled lines on a 33 MHz '486 PC. Converted to loopless *APL*, the execution time was reduced to <u>1.48</u> seconds. Using 'C' only cut the time to just below a second. In the mid-1980's, the same computation took over <u>5 minutes</u> on a 'high speed' scientific mainframe. With today's faster PC's, sacrificing ease of use (and programming) for small increases in execution speed is false economy. If a severe computational bottleneck <u>is</u> encountered, *APL* can call compiled code from C or FORTRAN.

documentation, and can use upper memory or disk as swap space to enlarge the available workspace size. A version of *LLAMA* to run under APL2000's *APL+DOS* (equivalent to Manugistic's *APL★PLUS II/386*) will be made available if there is a demand for it. *APL+DOS* uses fast 32-bit code, and can access extended memory to provide almost unlimited workspace size.

APLSE is a very powerful implementation of *APL*, but it lacks documentation. If you are new to *APL*, there are a number of useful files and tutorials located at a World Wide Web site at the University of Waterloo in Canada. The URL is ftp://archive.uwaterloo.ca/languages/apl/software-library/index.html. Under '1996 Additions', you will find not only *APLSE*, but a tutorial on its use. Jim Weigang at the University of Massachusetts at Amherst also has a WWW page at http://chilton.com/~jimw/. This includes information on his '*APL Notes*' book, which is an *APLSE*-based tutorial on *APL*. There is also an Internet 'newsgroup' devoted to the *APL* language and its dialects called 'comp.lang.apl' where you can ask questions about *APL* in general.

1.4 DEALING WITH THE *APL* CHARACTER SET

The documentation with *APLSE* includes instructions on how to set up the character set download programs for both the screen and a printer. This is best automated by using a batch file to call *APLSE*, and a sample batch file (APLSE.BAT) is included with *LLAMA* in the file LLAPLSE.ZIP along with some other useful items. Once you have *APLSE* up and running, you should follow the instructions for the *INITIAL* workspace, which allows you to print out the somewhat abbreviated manual. Chapter 1 includes a description of the *APL* keyboard layout, which will take a little getting used to. *APLSE* uses what APL2000 calls the 'Unified' keyboard layout, which leaves all standard characters in their normal locations. The *APL* keyboard of old was very confusing to many people, and this is a big improvement. If you are returning to APL after a long absence, this layout may come as a shock to you. It doesn't take long to un-learn the peculiar places APL used to keep things (like parentheses), and most of the special *APL* symbols are where they used to be, but are accessed with the Alt key. If you are using *APLSE*, the files in LLAPLSE.ZIP should be extracted and moved to the directory containing the APLSE executable file. LLAPLSE.ZIP includes a README.TXT file that explains briefly what each of the files does. The original *APL* character set only had a capitalized italic courier-like font for text, and all of the principle functions and variables in *MARTHA* and *LLAMA* use capital letters for their names, so it is usually best to set Caps-Lock on at the beginning of a session. As you have probably already noticed, a font similar to the original *APL* font is used in this manual to differentiate *APL* names, results etc. from regular text.

1.5 'LOCKED' FUNCTIONS

Inside some of the *LLAMA* workspaces (and all of the *MARTHA* workspaces) certain functions are 'locked'. This means that these functions can't be viewed, or more importantly, edited. In addition to protecting intellectual property, this reduces the chances that a user will shoot themself in the foot by editing code they don't fully understand. The original *MARTHA* code was written to run on machines with as little as 32K of memory. Not only was there no way to include comments in the code, a lot of subtle programming tricks were used to squeeze out every last unnecessary byte. For example, many *MARTHA* functions use absolute branching, and will cease to run if a single comment line is added at the top. Fortunately, the *MARTHA* code has been thoroughly tested and debugged, and there is no reason users should ever need to deal with its inner workings. Most of the newer *LLAMA* functions are heavily commented, and are unlocked to be used as examples or starting points for writing your own functions.

1.6 DOCUMENTATION FUNCTIONS

Each section of this manual is devoted to describing an *APL* 'workspace', which is just the *APL* way of collecting and organizing related functions. Each workspace corresponds to a DOS file, and (depending on the version of *APL*) will have a file extension of .aws, .ws or .w3. In order to provide the equivalent of on-line help, there are several documentation functions available in most of the *LLAMA* workspaces. These make use of 'public comments', which can be accessed even if a function is locked. Normal comments in APL begin with the 'lamp' symbol (ᴀ), while public comments begin with the pair of symbols 'ᴀ∇'. APL2000's *APLs* have a system function (one built into the interpreter) that can read these. There are three documentation functions that make use of this system:

SUMMARY: *SUMMARY* displays the first public-comment line of all of the 'main' functions in a workspace. A 'main' function is one expected to be executed directly by the user, and the function names are capitalized. There are many 'background' functions that are called by 'main' functions, and these are in lower case or begin with a special character sequence. The first comment line in most *LLAMA* functions contains a very brief description of what the function does.

Here is an abbreviated version of the result of running *SUMMARY* in the *LLMSDIM* workspace:

```
SUMMARY
 WSID:  8 LLMSDIM      10/14/96

 text←EXPLAIN fnname -- Returns all public comments from <fnname>
 out←KLᴀMSTRIP params -- Adds the Coupling 'KL' to the Output of MSTRIP
   |
 out←Ldummy WSFROMZ params -- Computes W/H and S/H from Zoe and Zoo
 Zoe,Zoo←ZFROMKL Zoe,KL -- Computes Zoo, Given Zoe and KL

   For list including background functions, use SUMMARYALL function
```

SUMMARYALL: As the *SUMMARY* function indicated above, the *SUMMARYALL* function displays both the main and background functions in a workspace. The result of running *SUMMARYALL* on the same workspace includes a number of background functions with lower case names:

```
        SUMMARYALL
 WSID:  8 LLMSDIM      10/14/96

 text←EXPLAIN fnname -- Returns all public comments from <fnname>
 out←KLᴀMSTRIP params -- Adds the Coupling 'KL' to the Output of MSTRIP
   |
 out←Ldummy WSFROMZ params -- Computes W/H and S/H from Zoe and Zoo
 Zoe,Zoo←ZFROMKL Zoe,KL -- Computes Zoo, Given Zoe and KL
 amat -- Background Fn for MSTRIP.  Creates Global 'a' and 'b' Matrices
 z←ci x -- Background Fn for MSTRIP.  Computes Cosine Integral
   |
 p←mgreen -- Background Fn for MSTRIP.  Green's Function Integration.
 p←mphi -- Background Fn for MSTRIP.  Computes PHI for ∈r=1 W/No Cover
```

EXPLAIN: The *EXPLAIN* function displays additional public comments from a function that describe what inputs are required, what the function does, and what outputs are produced. The name of the function to be 'explained' is passed as a right argument in the form of a string. For example,

4

here is what *EXPLAIN* has to say about the '*ALLPASS*' function found in the '*LLPHASE* ' workspace:

```
      EXPLAIN 'ALLPASS'

net←Zo ALLPASS Fo(Hz),Q -- Creates A 2nd Order All-Pass Network
Creates a MARTHA network description of the following circuit:
                        La
           |------nnnnn------|
           |                 |
           |   Ca        Ca  |
         O--o---||---o---||---o--O           Note: Q must be > 1
                    |
                   === Cb
                    |
         Zo →       ⊃             ← Zo
                    ⊃ Lb
                    ⊃
                    |
         O-----------o----------O

Circuit can be used to provide Group Delay equalization for filters.
REF: Williams, A.B., Taylor, F.J., 'Electronic Filter Design
Handbook' McGraw Hill, 1988, pp 7-4 to 7-6
```

In addition to these functions, all of the *LLAMA* workspaces contain a *DESCRIBE* variable that gives a brief overview of the functions is the workspace.

1.7 PROGRAMMING 'STYLE'

After years of working with *APL*, a number of programming conventions have been developed at Lincoln Laboratory that simplify using and maintaining a large number of workspaces and functions. To make the functions easier to use and to work with the public comment documentation functions above, most follow these general rules:

1) All functions that might normally be executed in immediate-mode by a user have capitalized names. Functions that are primarily called as background functions have lower-case names. Similar conventions are used for any global variables.

2) The first line of each function is a public comment (beginning with the symbols '*A∇*') indicating the general syntax (monadic, dyadic, etc.) and a <u>brief</u> description. This is what the user sees when the *SUMMARY* function is executed.

3) The next section of the function consists of public comments giving a detailed description of what it does, what the inputs are, range limitations, and references. This is what is listed when the user runs the *EXPLAIN* function. This is generally much more readable if you type it in lower-case (i.e. normal) text instead of capitalized *APL* text.

4) The first few lines of actual code check to see if the number of inputs in the right argument is correct. If not, the function prompts the user for the proper inputs, or exits with a polite warning. This means that if you remember what function you want, but don't quite remember the input

5

format, you can call it with an empty vector (' ') and get prompted. A check is made for the existence of the left argument, to allow prompting without requiring prior knowledge of whether the function is monadic or dyadic.

5) Once the correct number of inputs is obtained, they are checked against any range limitations the function may have. If an error is detected, the function either: a) branches to zero and produces an error message, while returning a result any calling function can recognize as erroneous, or b) uses the $\square ERROR$ function to send an error message and terminate with everything pending in the state-indicator. Option a) is a bit cleaner, but requires care when writing higher level calling functions.

1.8 LIBRARIES AND SETTING UP *APLSE*

APL uses the term 'library' to describe a group of workspaces. On personal computers, the *APL* libraries correspond to sub-directories. Internal to *APL*, libraries are designated by numbers, which saves a lot of typing, but requires the user to set up a library file to let *APL* know where to find things. In APL2000's *APL*, this is typically done with a file called LIBS.APL. If you are using one of the commercial *APL*'s, the documentation explains how to set this up. For those using *APLSE*, there are two files included in LLAPLSE.ZIP that you can use to get started:

1) The first is called LIBS.APL file. This is just an ASCII text file, and contains the following:

```
!--------- Library Definitions for APLSE
lib 0 = A:
lib 1 = B:
lib 2 = C:\APLSE
lib 3 = C:\APLSE\MYWS
lib 7 = C:\APLSE\MARTHA
lib 8 = C:\APLSE\LLAMA
!NOTE:  LLAMA workspaces must be in Library #8
```

As you can see, the library numbers do not have to be in sequence.

2) The second file is called CONFIG.APL. This is an optional file used to configure your *APL* system, and has a lot of useful features. Unfortunately, the documentation that comes with *APLSE* makes no mention of it. CONFIG.APL is also just an ASCII file containing:

```
!          CONFIG.APL File for APLSE
! Other parameters exist beyond those shown here; ! marks comments
! Adjust directories and uncomment as desired
!
!----- Environment Information
d=15                ! Extra []LIBD libraries after startup
printerport=10  ! Printer output via APLPRINT.COM or APLPS.COM drivers
translation=2   ! Character translation for printer; 2 = ASCII
config=C:\APLSE\LIBS.APL    ! Library definitions
!
!----- Session Manager Options
capslock=ON    ! ON useful for Text keyboard; irrelevant for APL keyboard
numlock=OFF    ! OFF means numeric pad will provide cursor movements
insert=ON      ! Keyboard insert or replace mode
dragdown=ON    ! ON -> re-entered lines of input copied to end of session
screenmem=30   ! Total session manager & editor memory
initWs= '3 INITWS'    ! Start-Up Workspace
```

There are three entries of particular note. About halfway down is the line that tells *APL* where to find the library file. The last two lines set how much memory the editor gets to work with, and the last line sets up a start-up workspace, which is a good place to park your printer character-set download function and any other *APL* operations you need to run at the beginning of a session.

Both files should be located in the directory containing the *APLSE* executable. You can edit these files to match your own directory structure, with one CRITICAL exception. A large number of the *LLAMA* workspaces need to know where the *LLAMA* files are stored, and have been 'hardwired' to look in Library 8. If you can't follow this convention because you have an existing *APL* system with Library 8 in use, there are a number of changes that will need to be made in the *LLAMA* workspaces. These are all covered in the appropriate Sections of this manual.

The remainder of this manual will go into detail on each of the *LLAMA* workspaces; how they are used and what they do.

1.9 ACKNOWLEDGMENTS

As was mentioned at the beginning, *LLAMA* is the result of over 20 years of programming at MIT Lincoln Laboratory. To the extent that it could be traced or remembered, credit for individual functions is given in each section. Many of these functions were originated long before the author of this manual arrived at Lincoln Lab, and many of the original authors of the mainframe *APL* have either retired or left for other jobs. When much of this code was rescued from the Lab's mainframe, it was quite a mix of programming styles and the level of documentation varied wildly, including who had done what. Every attempt has been made to give credit where it is due, and any omissions are solely due to lack of information. Any function not specifically mentioned is probably the author's own work.

Particular thanks for much of what is here go to two people. First and foremost is Prof. Paul Penfield, Jr. at MIT, whose *MARTHA* started it all. His continued work produced a number of essential *LLAMA* functions, and his support was invaluable in rescuing *MARTHA* from the mainframe and getting it working on PC's. The second person is David Hodsdon, who re-introduced the author to *MARTHA* when he first arrived at Lincoln Lab, and who wrote a large number of the original functions on which *LLAMA* is based.

I would also like to thank Mark Fishman for having the patience to carefully proofread the final version of this manual. He efforts have made this a much more polished document than it might have been.

1.10 REFERENCES:

1.1   Penfield, Paul, Jr. "*MARTHA* Users Manual," MIT Press, Cambridge, MA, USA, 1971

1.2   Penfield, Paul, Jr. "*MARTHA* Users Manual, 1973 Addendum," Dept. of Electrical Engineering, MIT, Cambridge, MA, USA, 1974

2. *LLUTILTY* WORKSPACE

UTILITIES FOR USE WITH *MARTHA* AND *LLAMA*

2.1 INTRODUCTION

This workspace contains a collection of small functions that are useful for managing and maintaining your workspaces, along with a few functions that just perform useful or common tasks. All of the functions have been documented using the 'public-comment' functions described briefly in the introduction. All three of the documentation functions (*'EXPLAIN'*, *'SUMMARY'* and *'SUMMARYALL'*) are included in *LLUTILTY*, partly for use with the workspace, and partly as a convenient place to copy them from when you are setting up your own workspaces. For completeness, here are the results of running *EXPLAIN* on all three functions:

```
    EXPLAIN 'EXPLAIN'

text←EXPLAIN fnname -- Returns all public comments from <fnname>
Modified to show ALL public comments in unlocked fns, not just  lines
at top.
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T


    EXPLAIN 'SUMMARY'

SUMMARY -- Display a short description of MAIN fns in active workspace
Does not display background fns beginning with lower case letters,
or 'V_', which signifies a VDI graphics function.  To see all fns,
use 'SUMMARYALL'.
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T


    EXPLAIN 'SUMMARYALL'

SUMMARYALL -- Display a short description of ALL fns in active workspace
To skip display of background fns, use 'SUMMARY' fn.
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T
```

The purpose of most of the functions is fairly obvious from the *SUMMARYALL* and *EXPLAIN* results. The remaining functions in *LLUTILTY* have been loosely grouped into seven different categories, and this section describes their use and operation in detail. Copies of the *EXPLAIN* listing for all of the functions are included (minus the version/copyright line to save some ink).

2.2 THE '*LEAVE*' WORKSPACE COPYING SYSTEM

The functions '*LEAVE*', '*COPYΔWS*', and '*ΔNL*' work together to provide an automated means of copying in a major workspace (like *LLMARTHA*) when a workspace is loaded, and then erasing it when you are done. This saves a considerable amount of disk space because you don't keep extra copies of common functions in every workspace you use. It is also much easier to work with, and maintain, a workspace that contains only the functions unique to its purpose.

Many of the *LLAMA* workspaces are already set up to use the '*LEAVE*' system to copy in '*LLMARTHA*'. It is important to understand how the system works in case something goes haywire. The '*EXPLAIN*' results for '*LEAVE*' cover the basics:

```
LEAVE -- Erases LLPLOT and LLMARTHA functions and variables, then saves WS
Uses 'COPYΔWS' fn along with the background variables 'NL2Δwsid' and
'NL3Δwsid' to keep track of what to save or erase.  These variables
are created by the 'ΔNL' function, which should be executed BEFORE
either LLPLOT or LLMARTHA are copied into the active WS.

SETUP:  (1) Create a WS you would like to use with LLPLOT or LLMARTHA.
        (2) Run ΔNL.  This creates the 'NL2/3Δ' variables
        (3) Execute: 1 COPYΔWS '8 LLPLOT' or '8 LLMARTHA'.  This PCOPY's
            objects from the appropriate WS.
        (4) When done, run LEAVE.  This erases fns and vars from LLPLOT
            and LLMARTHA, and saves the workspace.
        (5) Add 1 COPYΔWS '8 LLPLOT' or '8 LLMARTHA' to your Latent Expression
            (□LX) so they will be copied automatically on Startup.
        (6) Periodically, run ΔNL after executing LEAVE, then save the
            WS.  This updates the object lists.

NOTE: Other WS's can be added to the process as long as they contain
their own 'NL2/3Δ' variables created by running ΔNL.
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T
```

In step (1), you create and name a workspace, and copy in '*LEAVE*', '*ΔNL*' and '*COPYΔWS*'. You can then create any variables or functions with which you wish to work. When you execute '*ΔNL*' in step (2), it creates two variables in the workspace, $NL2\Delta wsname$ and $NL3\Delta wsname$. These contain a list of the variables and functions in the workspace, respectively.

At this point, you can load in either *LLPLOT* or *LLMARTHA* using the *COPYΔWS* function. It is assumed that both are in Library 8. If you set up your system with another Library for *LLAMA*, you will have to change the right argument of *COPYΔWS* in any workspaces that use this system. When *COPYΔWS* is run with a left argument of '1', it does a protective copy of the workspace. This means that none of the *LLMARTHA* or *LLPLOT* objects will overwrite anything with the same name. Both of these workspaces have their own *ΔNL* variables, which will be copied into your workspace. You can now do whatever work is required with the combined workspace.

When you are done, you execute the *LEAVE* function. This does two things: it carefully erases ONLY those functions that were copied in step (3), and it saves the workspace.

In order to automatically copy in either *LLMARTHA* or *LLPLOT* when a workspace is loaded, in step (5), you put the appropriate *COPYΔWS* command into the Latent Expression (□LX) of your workspace. When a workspace is loaded, any string assigned to □LX gets executed, so you could just type:

```
□LX←'1 COPYΔWS ''8 LLMARTHA''' <Enter>
```

It is frequently easier to have the name of a 'start-up' function assigned to □LX. In all of the *LLAMA* workspaces that use this system, the function is '*STARt*'. This is where you would change the

library number if required, and it provides a convenient place to initialize any variables, display a title etc. Make sure you save the workspace after you have assigned a string to $\square LX$.

Step (6) is optional, but provides some insurance in case something goes wrong. Any time you run $\triangle NL$, it updates the two $NL2/3$ variables. It is VERY important that you DON'T run $\triangle NL$ while $LLMARTHA$ or $LLPLOT$ are still in the workspace. If you do, it will assume that everything belongs in your workspace, and running $LEAVE$ will have no effect.

One common issue that arises is what to do if you wish to copy an existing workspace. For example, if you want to design some filters for a project, it would be useful to have a copy of the $LLFILTER$ workspace dedicated to that project without cluttering up the original workspace. To do this, you would first load the $LLFILTER$ workspace, and then execute $LEAVE$. You can then rename the workspace and save it. At this point, everything will behave normally when you load the workspace, even though it contains the variables $NL2\triangle LLFILTER$ and $NL3\triangle LLFILTER$, and the title will still claim it's $LLFILTER$. The only problem arises if you execute $\triangle NL$ at this point. It will create a NEW set of $NL2/3$ variables, and $COPY\triangle WS$ will produce an error message about too many variables. All you need to do is erase the old $NL2/3\triangle LLFILTER$ variables, and everything will function normally.

One additional trick is used with this system. When $LLPLOT$ or $LLMARTHA$ is originally $PCOPIED$ into a workspace, all of the background variables should come with them. However, the titles and plot variables are frequently customized for a particular workspace, and you don't want to lose them every time you run $LEAVE$. In order to do this, the names of these variables have been removed from the $NL2/3$ variables in $LLPLOT$ and $LLMARTHA$. This way they are brought in the first time, but they are never erased. In order for this to work properly, $\triangle NL$ should NOT be run on the original $LLPLOT$ and $LLMARTHA$ workspaces, because it will place the plot parameter variables back into the $NL2/3$ variables.

Here are the $EXPLAIN$ listings for both $COPY\triangle WS$ and $\triangle NL$:

```
mode COPYΔWS copyws -- Automates copying/erasing workspaces (see LEAVE)
If 'mode'=1, and 'copyws' isn't already present, performs a PCOPY.
If 'mode'=0, and 'copyws' is present, erases only objects copied from
'copyws'.  Uses background variables 'NL2Δwsname', 'NL3Δwsname',
'NL2Δcopyws', and 'NL3Δcopyws' created by the 'ΔNL' function.  Library
numbers are required to copy, but not to erase.

ΔNL -- Creates 'NL' variables for use with COPYΔWS function.
Execute when workspace contains ONLY those items you wish to save.
```

2.3 WORKSPACE MAINTENANCE FUNCTIONS

Once you have used various workspaces for a while, they tend to fill up with a lot of old variables, copies of functions etc. If you want to use one of these workspaces for a new project, some 'housecleaning' is frequently in order. $LLUTILTY$ includes a number of functions to help in this task.

The function '$DIFF$' allows you to compare functions in two different workspaces. This is useful if you've inadvertently ended up with two functions of the same name, or are trying to sort out two different revisions of a function. You can also compare entire workspaces by omitting the function list.

```
Text←[fnname(s)] DIFF wsname -- Compares local fns with those in 'wsname'
Compare function 'fnname' with function of same name (all functions
if 'fnname' blank or absent) and report any differences.  'fnname'
can also be a matrix of names.
```

While developing a new function, it is often useful to have a printed copy on which to scribble. Short functions can be printed out using the Shift-Print Screen key combination, but a better approach is to use the '*PRINTL*' function. *PRINTL* prints out listings of any functions or variables in its argument to an HP LaserJet compatible printer. Each page includes a header with additional useful information, as described in the *EXPLAIN* listing below:

```
PRINTL Name(s) -- Prints Listing of Named Objects to HP Laserjets
Objects can be a mix of either variables or functions.  Name(s) can
be either a single name, a space or comma delimited list, or a matrix.
Each page has a header with the workspace name, the date and page #.
Based on Paul Penfield's 'DOC' function.
```

Once you have slaved over developing a workspace for months, and you <u>finally</u> have it all working the way you want, it's a good idea to keep a printed copy of the complete workspace around for future reference. The function '*DOC*' automates this process. *DOC* prints out a listing not only of the functions, but all the variables in a workspace. It's not a bad idea to run *VERLIST* (see below) first to make sure you don't have any enormous variables that will take 30 pages to print out. *DOC* is set up to call the function '*GLOBAL*' to give a complete list of all of the global variables in the workspace. If *LLUTILTY* is not located in Library 8, *DOC* won't run unless you either edit the location or omit running *GLOBAL* (see the *EXPLAIN* listing below). This function only works with HP LaserJet compatible printers, and includes a number of subtle programming tricks to avoid interactions between *DOC* and the workspace in which it is running. For example, there are no labels used for branching, because these would be treated as variables within the workspace. They would be listed as such, possibly masking the existence of a real variable of the same name. The function is provided unlocked so the Library location of *GLOBAL* can be changed, but further modifications are strongly discouraged.

```
DOC Title -- Prints detailed workspace listing to HP Laserjet printers
Document workspace non-destructively.  'Title' appears as WS name in
header on each page.  Suspended fns are OK.  Includes call to fn
GLOBAL to list all global variables in fns.  If title string begins
with '/', call to GLOBAL is omitted.  'DOC' may require editing to
give correct location of GLOBAL (see ⎕PCOPY in line [13]).  If Title
= LLUTILTY, 'DOC' is included in listings.  DOC will crash on any
workspace containing the Laserjet download variables 'dlHPLJ' and
'dlHPLJL'.  Program also doesn't correctly list vars or fns named 't',
'u', 'v', 'w', 'x', 'y', or 'z'.
```

Although generally to be avoided, global variables are occasionally useful. If you need to clean up the variables in a workspace, some of them may be required as global variables for important functions. As mentioned above, the function *GLOBAL* scans a function and identifies all of the global variables it uses. *GLOBAL* can also be used to scan an entire workspace for occurrences of global variables.

```
Text←GLOBAL fnname(s) -- Lists all global names in fnname(s)
Also lists fn lines containing '≄' (execute) which might contain
additional names.  'fnname' can be a matrix of names.  If it is an
empty vector, all of the fns in the WS will be checked.  GLOBAL
doesn't recognize functions named 'y' or 'z'.
```

There are two functions for screening the list of variables in a workspace; '*LISTVARS*' and '*FLUSHVARS*'. *LISTVARS* gives a quick summary of all the variables in a workspace, and *FLUSHVARS* allows you to go through them and delete any that are no longer of interest.

```
LISTVARS -- Lists and briefly describes variables in WS
Lists all of the variables in a workspace, along with their type
(numeric or character), the 1st element or 10 characters, and their
shape (size).
```

```
FLUSHVARS -- Lists, describes and allows deletion of vars in WS
Lists all of the variables in a workspace, along with their type
(numeric or character), the 1st element or 10 characters, and their
shape (size).  After displaying the information on a given variable,
the user can erase it, exit the function, or go on to the next var.
```

All of the unlocked functions in *LLAMA* have a version string placed on the last line as a public comment. This is a programming convention that is HIGHLY recommended for any functions you may write. All of the *LLAMA* functions are being released initially as version 1.0, but bug-fixes and updates in the future will have different version numbers. If you want to check that you are running the latest version, the '*VERLIST*' function displays the version string from all of the functions in a workspace.

```
VERLIST -- Displays the Names and Version Notes of Fns in Workspace
Assumes version note is a public comment (beginning with 'ዘ∇') on
the last line of a function, beginning with the characters ' V'
```

APL uses something called a 'symbol table' to keep track of all of the objects (functions and variables) in a workspace. For speed reasons, older versions of *APL* use a fixed sized table and don't do 'garbage collection' on the symbol table. Eventually, as you move variables around and run more functions, the symbol table can fill up. At this point, the workspace will stop working until the symbol table is cleared or a garbage collection is done. Some APL's have '*□POKE*' commands that will invoke a garbage collection. Another approach is to copy the workspace into a clear workspace, but then it must be renamed and the Latent Expression (□LX) restored. The function '*RESET*' does this automatically, and increases the symbol table size in the process. The larger symbol table size is frequently necessary when working with *LLMARTHA* because of the large number of functions included in the workspace.

```
RESET -- Clears symbol table, then resaves WS with old name and □LX
Stores the entire sequence of commands required to clear the symbol
table in the keyboard buffer, then executes.  Sets symbol table to
1024 elements (APL2000 default is 512).
```

It is possible (and recommended) to write well commented *APL* that can be easily modified or embellished in the future. Compact and cryptic code is easy to write in *APL*, but it is usually indecipherable even to its author within 6 months. Unfortunately, comments take up space. If you find

that you are tight for memory, you can remove all but the public comments (used for the *EXPLAIN* and *SUMMARY* system) using the '*PDECOM*' function. The *LLPLOT* workspace contains a LOT of comments, and running *PDECOM* on a <u>COPY</u> of *LLPLOT* can free up a lot of space. PLEASE don't do this to your only copy of a workspace! Like locking a function, there is no way of reversing this process.

```
ok?←PDECOM fnname -- Strips non-public comments from unlocked fn(s) 'fnname'
Result is ¯1 for non- or locked fn; 1 if successful
If 'fnname' is a matrix of fnnames, PDECOM loops through them
```

The '*WHEREIS*' function helps solve two common problems. The first is when a variable in your workspace reappears despite frequent deletions. This means that someone forgot to make the variable local, and it's being re-created as a global variable every time <u>some</u> function is run. *WHEREIS* lists every function in the workspace that uses the variable, and shows if the variable is localized in line [0]. The second problem occurs when you want to erase a function, but aren't sure if it's crucial to another function you need. *WHEREIS* makes a list of functions that might be calling the one in question.

```
Text←WHEREIS String -- Locate all occurences of String in all fns in WS
Returns fnname[Lineno(s)] for each occurrence.  String can be a text
string, a variable, or a fn name.  WHEREIS indicates if String was
found as a comment, string, or object name.
```

2.4 TIMING FUNCTION

Although modern PC's are quite fast, some complex simulations can take a while to execute. It is handy to have a way of timing such things, and the function '*MEAS∆T*' was developed for this purpose.

```
MEAS∆T st -- Displays time between executing 'MEAS∆T 0' and 'MEAS∆T 1'
Used to display elapsed time by executing just before and just after
a given operation.  Accuracy is limited by system clock.  Uses the
global var 'to', which is erased when 'st'≠0.
```

2.5 NATIVE FILE FUNCTIONS

APL uses a special file format for storing information, and refers to standard DOS files as 'Native Files'. It is easy to use the editor to move text files in and out of *APL*, but it is often useful to import or export numerical files as well. The functions '*ASCIIOUT*' and '*ASCIIGET*' work with numerical data matrixes, and automatically translate the *APL* high minus sign (¯) to work with other applications.

```
'File' ASCIIOUT Data -- Creates an ASCII DOS File of a Data Matrix
'File' is any valid DOS file name, and will automatically be
assigned an extension of '.asc'.  NOTE: If you get 'FILE NAME ERROR',
check to make sure that the filename doesn't already exist.

Vect←ASCIIGET File  -- Converts ASCII Text Files to APL Numeric Vectors
File ← '(Drive):\(Path)\(Filename).(Extension)'.  Source file must
contain only numeric data.  Result is a vector containing the
elements in the source file.  Has been tested with Lotus spreadsheet
print file.
```

The function 'S∆PARAM' is more specialized, in that it takes a 9 column matrix (one for frequency and four S-Parameters in magnitude/angle format) and creates a 'Touchstone' compatible ASCII file. This format can be read by a number of commercial RF CAD packages.

```
'filename' S∆PARAM Mat -- Creates a Touchstone 2 Port S-Parameter File
The filename must be a string.  The filename should include the
path BUT not the file ext. '.S2P'.  This is added by the S∆PARAM fn.
Mat must be a 9 column matrix.  Column 1 = Freq in MHz, remaining
columns are pairs of mag/ang S-parameters in order: S11, S21, S12, S22
NOTE: If you get 'FILE NAME ERROR', check to make sure that the
filename doesn't already exist.
```

2.6 WINDOWS BITMAP REPAIR FUNCTION

One of the nicer features of working under Microsoft Windows is the ability to move information easily from one application to another. Although it has some quirks, there is a procedure for getting plots made with the *APL* plotting functions into a Windows compatible bitmap. All of the plots in this manual were created using this system. The procedure is as follows:

1) Create the desired plot in a <u>full screen</u> session of *APL*, running under Windows.

2) Press Alt-PrintScreen. This copies the screen into the clipboard. Press any key to get back to *APL* from the graphics screen, then press Alt-Enter to reduce the *APL* window.

3) Open the Windows Paintbrush application. Under 'Options', select 'Image Attributes'. Set 'Width' to 5.35 and 'Height' to 4 (units should be inches). The options 'stick', and you should only have to do this once. If needed, maximize so Paintbrush is running in full-screen mode.

4) Using the 'Edit' menu, select 'Paste'. After a second or so, your plot will appear on the screen. Unfortunately, on many systems, the background will be black. Using the 'File - Save As' command, save the image as a <u>16 color</u> bitmap (.BMP) file using the 'Save File as Type' menu.

5) *LLUTILTY* contains the function '*FIXBMP*', which corrects the color problem. The *EXPLAIN* listing for *FIXBMP* is shown below:

```
OK?←FIXBMP File -- Corrects Black Background in APL Plot Bitmaps
File ← '(Drive):\(Path)\(Filename).(Extension)'.  Source file must
be a 16 color bitmap file (.BMP) from Windows Paintbrush.
Function returns a '1' if the operation was successful, a '0'
otherwise.
```

Get back into *APL*, and run *FIXBMP* on the file you saved in Paintbrush. This should restore the plot background to white, and the foreground color of the axes and labels to black.

6) If you are planning on using the plot in a word-processor document and only need a black and white plot, you can save considerable disk space by converting the .BMP file to monochrome. This can be done by reloading the file into Paintbrush, and then using 'Save As' to convert it to monochrome with the 'Save File as Type' menu. This will reduce the file size by a factor of four. A typical monochrome .BMP file created this way will be about 40 Kbytes.

15

Below are two example plots. The first (Fig. 2.1) was scanned in. The second plot (Fig. 2.2) was captured using the new approach. Although the process may sound a bit involved, it's not too bad once you've got things set up and have had a little practice.
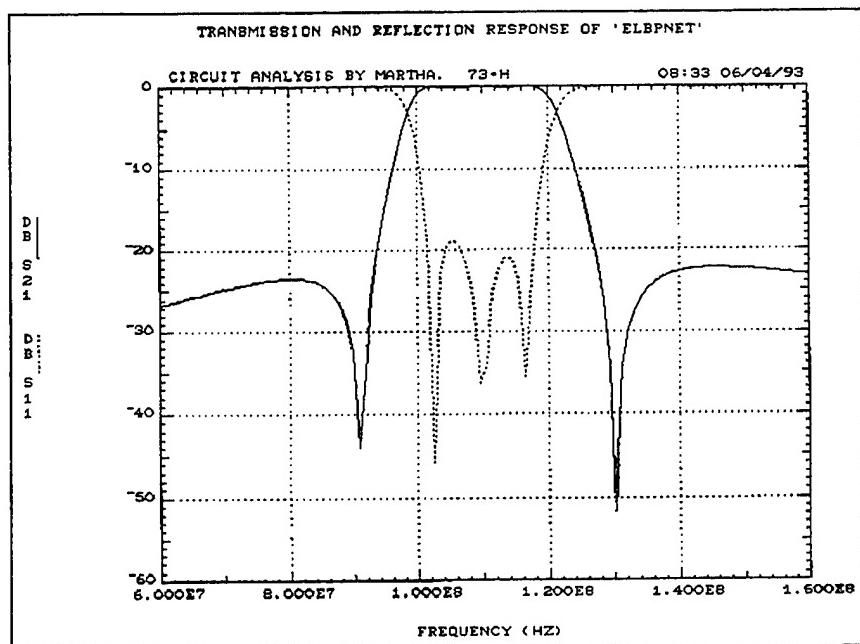

Figure 2.1. Typical scanned image
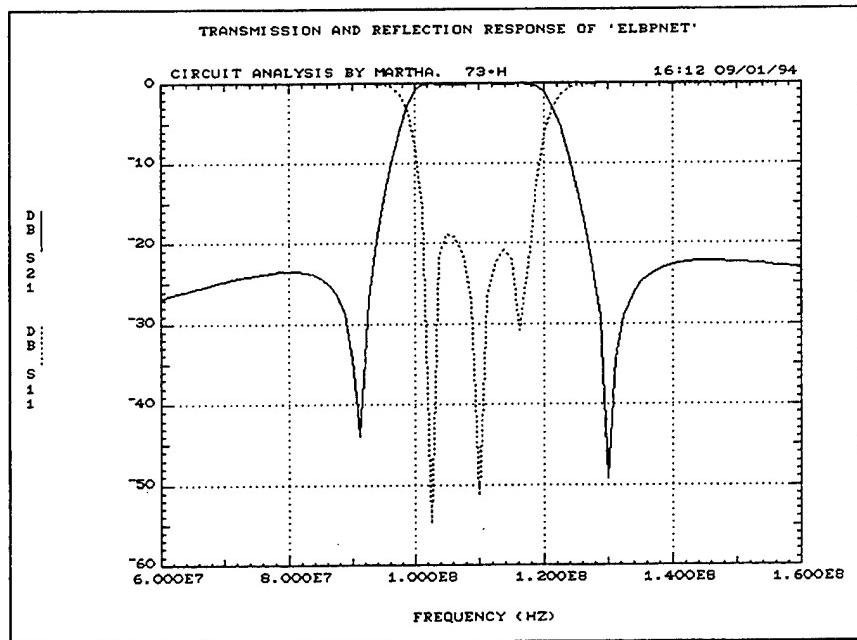
Figure 2.2. Plot captured from screen

16

2.7 CHARACTER VECTOR/MATRIX FUNCTIONS

There are two functions that allow you to convert character variables back and forth between a vector form (with embedded returns) and a matrix. The matrix form can be wasteful of space, because all short lines are padded with spaces to the length of the maximum line, but it is sometimes easier to work with in a program. The function *M2V* converts a matrix to a vector, and *V2M* converts the other way.

```
CharVect←M2V CharMat -- Converts a Text Matrix to a Vector

CharMat←V2M CharVect -- Converts a Text Vector to a Matrix
```

2.8 MISCELLANEOUS FUNCTIONS

There are four other functions in *LLUTILTY* that are useful for specific tasks. If you just want a quick print-out of a function or variable, you can use Shift-Print Screen, or Ctrl-Print Screen to send the screen contents to the printer. Unfortunately, you need to issue a form feed to get the last page to come out, or take the printer off-line and eject it manually. The function '*PF1FF*' takes care of this by setting the keyboard function key 'F1' to issue a form-feed to the printer. This is a handy function to run at the beginning of any *APL* session, and can be run automatically in your start-up workspace so that it is always ready to go.

```
PF1FF -- Sets Function Key F1 to send a form-feed to printer
```

The next two functions are used to create time-stamps and workspace identification for any functions that might need them.

```
String←DATE -- Returns time and date in form; 12:01 HOURS, MAY 04, 1992

String←WSDATE -- Returns WSID, time, and date
```

The last function, '*efmt*', takes care of a quirk of APL2000's exponential number formatting function. The original mainframe system would format a number like 0.0005 as $5E^-4$, which is perfectly fine. Unfortunately, the PC *APL* would produce $5E^-004$, which is not only a bit less esthetic, but also raised havoc with the alignment and spacing of some output tables because of all of the extra zeroes. Rather than redesign all the tables and live with all those extra zeroes, *efmt* was written to duplicate the old mainframe result.

```
Z←D efmt X -- 'Clean' exponential format fn.   'D ⍕ X' w/o extra zeros
Removes extra zeros from exponent for shorter labels.  Also adds '0' in
front of decimal formats less than magnitude 1.  NOTE: At this time,
this function can't handle vectors or matrices of numbers.
```

2.9 ACKNOWLEDGMENTS

The *DOC*, *DIFF*, *GLOBAL*, *M2V*, *V2M* and *WHEREIS* functions were all originally written by Paul Penfield, Jr. The *LEAVE* system and its associated functions were developed by David Hodsdon, as was the *MEAS*Δ*T* function. Mark Stevens originated the *ASCIIGET* and *ASCIIOUT* functions.

The *EXPLAIN*, *SUMMARY* and *SUMMARYALL* functions are based largely on similar functions used by APL2000 with their workspaces.

3. *LLPLOT* AND *MARTHAP* WORKSPACES

XY PLOTTING WORKSPACES

3.1 INTRODUCTION

This is a description of the *APL* plotting workspaces developed at MIT Lincoln Laboratory. They use the '□*G*' APL2000 PC graphics to implement a general purpose XY plotting package, including Log X, Log Y and Log XY with or without autoscaling. The general operation is based on an old Lincoln Laboratory mainframe workspace called '*TEKPLOT*', but with a number of embellishments and additional features. Hardcopy output can be obtained on Epson type dot-matrix printers, HP LaserJet or DeskJet printers. There are two workspaces supplied with *LLAMA*; '*LLPLOT*' and '*MARTHAP* '. *LLPLOT* is a general purpose plotting package suitable for a variety of applications. It is supplied with all of the comments intact for the curious. The *MARTHA* version, *MARTHAP*, has had all of the non-public comments removed in order to save space. This section starts with a general description of the plotting operation, with a section at the end that details the differences in *MARTHAP*. Most of this information is also contained in the workspace '*HELPPLOT*' if you need help while in *APL*.

3.2 FIRST TIME USE

Before doing a plot, you will need to set up the video adapter and printer selections. Start by running '*DEFAULT*', which will strip out any existing selections. Then run '*ginit*' and '*print*'. These will prompt you with the currently supported hardware choices. If you don't see what you need, you will have to add the appropriate codes by consulting the APL2000 Reference Manual. If you try to run *PLOT* without this step, you will be prompted when the functions are called the first time. WARNING: If you obtained this software from someone who has already been through the setup procedure, the flags are set and the program will try to talk to your hardware accordingly. You have been warned! It is remotely possible that HARDWARE DAMAGE will result. HINT: If you give a copy to someone, run '*DEFAULT*' before you give it to them to remove your flags.

3.3 BASIC FUNCTIONS

Functions typically called by the user have upper-case names, background functions have lower case names. Everything has been setup to work with the '*EXPLAIN*' documentation system if you need help working with a particular function. There are four types of basic user functions: plotting functions, formatting functions, setup functions, and help functions.

PLOTTING FUNCTIONS: There are 4 basic plotting functions:

PLOT, which does linear XY plots.
XYLOGPLOT, which does Log-Log plots.
XLOGPLOT, which does Linear Y vs Log X plots.
YLOGPLOT, which does Log Y vs Linear X plots.

All four functions are monadic, and can handle a variety of right arguments. The results for different shape right arguments inputs are as follows:

1) Vector, or single row or column array: The output is a plot with the input on the vertical axis, and the indices of the vector as the horizontal axis.

2) Two dimensional array, or single plane 3 dimensional array: The last column is the horizontal variable, and each additional column to the left is a separate vertical variable. When plotted, each trace can designated with a different color.

3) Multiplane array: Each plane is interpreted as a separate set of curves, following the rules for 2 dimensional or single plane arrays. The last column of each plane represents a different horizontal variable. Note: It is possible to plot multiple curves with different numbers of points by padding the smaller arrays with multiple copies of the array's end-points.

The color and linetype of each different plot variable are selected in rotation from the color sequence set in the function *RESTORE*, and the linetype sequence set by the *PLOTSEQ* function. If the *POINTS* function is used to plot individual data points, the markers used are set by the '*mark*' background function.

Below is a sample plot showing multiple variables plotted with lines and markers, as well as a multi-line title and X-axis label.
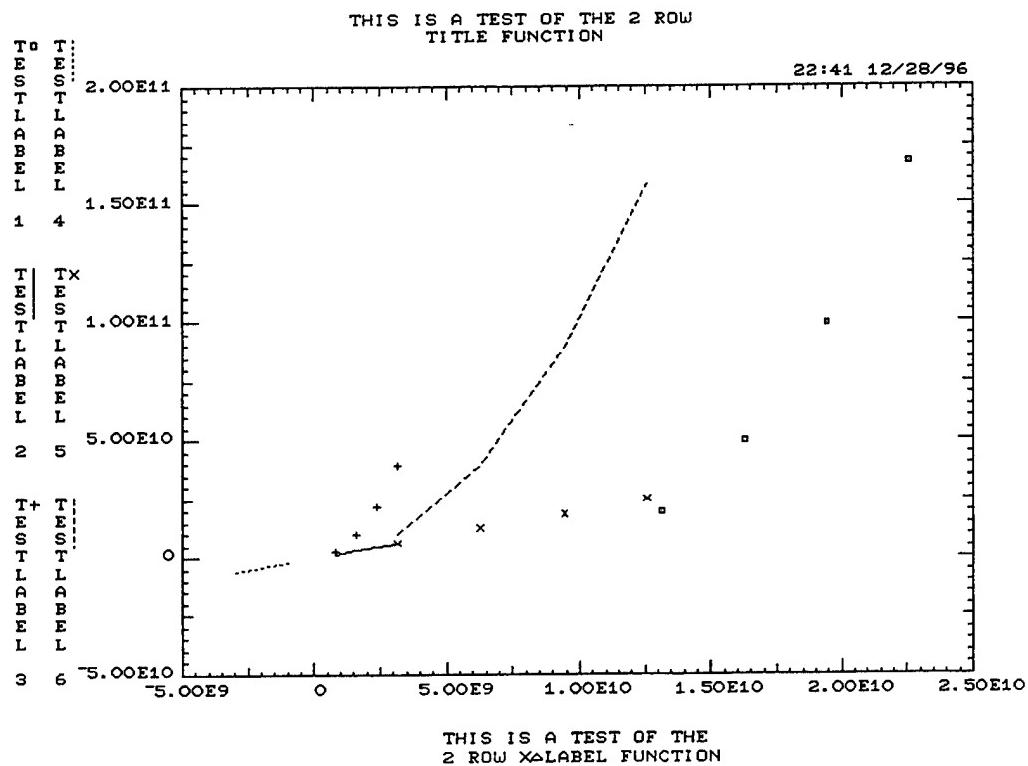

Figure 3.1. Sample plot with multiple variables

3.4 MARKING & LABELING WITH THE MOUSE

The various plotting functions support markers and annotation with a mouse. The sequence of marking and labeling options is as follows:

MODE:Pause (plot is on screen, no mouse readout at lower left)

    Mouse Buttons: Left - enter Marker mode
                         Right - enter Text Label mode
                         Both - Return to text mode

MODE:Marker (mouse readout at lower left)

    Mouse Buttons: Left - Place mark at mouse pointer location & enter Label Placement mode
                         Right - same as Left
                         Both - Return to Pause mode without marking

MODE:Text Label (mouse readout at lower left)

    Mouse Buttons: Left - Place mark at pointer location, prompts for text entry
                                  (CR terminates text and enters Text Placement mode)
                         Right - same as Left
                         Both - Return to Pause mode without marking

MODE:Label Placement (mouse readout at lower left is frozen)

    Mouse Buttons: Left - Place mark and coordinate label at mouse pointer location, return to
                                  Pause mode
                         Right - Place mark and coordinate beneath last label to make a table,
                                  return to Pause
                         Both - Return to Pause mode without labeling

MODE:Text Placement (Text label frozen at lower left)

    Mouse Buttons: Left - Place mark and text at mouse pointer location, return to Pause mode
                         Right - Place mark and text beneath last label to make a table, return to
                                  Pause mode
                         Both - Return to Pause mode without placing text

3.5 PRINTING

Pressing <shift> P while in Pause mode (no data or text at lower left) will produce a screen dump to your printer. Once the hardcopy process is complete, the program returns to Pause mode, rather than leaving graphics. To exit the plotting function, press any other key, or both mouse buttons while in Pause mode.

3.6 FORMATTING FUNCTIONS

There are 2 formatting functions, used to construct the right argument for the plotting functions from various types of inputs. Both functions are dyadic. The functions are:

VS: This is used to associate a horizontal variable with one or more vertical variables. The syntax is:

```
PLOT Y VS X
```

In the simple case of X and Y being 2 equal length vectors, this will produce a plot with Y as the vertical variable, and X as the horizontal variable. Y can also be an array, with each column representing a different set of points to be plotted against X. X can be either a vector or a single row or column array, but it must have the same number of elements as Y, or as Y has rows. The result of VS is a three dimension array of depth 1, with X being the last column.

AND: This is used to associate multiple vertical variables with a horizontal variable, or multiple XY data sets for display in a single plot. The syntax is:

```
PLOT A AND B, or

PLOT A AND B VS X, or

PLOT (A VS X1) AND B VS X2
```

In the first two cases, A can be a vector, a single row or column array, or a multi-column 2 dimensional array, representing one or more vertical variables. B can be a vector, a single row or column array, a multi-column 2 dimensional array, or a 3 dimensional array. If B represents only a single list of data, A and B will be plotted against a horizontal axis of indices ($\iota\rho$, B). If B is 2 or 3 dimensions, it is assumed to represent one or more vertical variables with the last column as the horizontal variable. The AND function appends the left argument to the right argument to produce a three dimensional array of depth 1, with the last column being the horizontal variable.

In the second case, (A VS $X1$) will be a 3 dimensional array of depth 1. This signifies to the AND function that the left argument represents a set of XY data, and not just more vertical variables. In this case, AND will produce a 3 dimensional array with one plane for each set of horizontal variables. If the left argument has a different number of rows or columns from the right argument, the array with the smallest dimension will be padded with multiple copies of its endpoint(s) to fill it out to conform to the larger array. These will not actually be plotted, but they won't upset the autoscaling this way.

3.7 SETUP FUNCTIONS

There are a number of functions that allow you to control and modify your plots:

$AUTOSCALE$: This turns autoscaling on and off. The default is 'on'. It can be used with the dummy function 'SET' and the variables 'ON' (= 1) and 'OFF' (= 0) i.e. :

```
SET AUTOSCALE ON
```

$DEFAULT$: This restores everything to a known state, including resetting the video adapter and printer selections to 'none'. Restores the axis labels and title to echo the variable name ($TITLE\triangle='TITLE\triangle'$ etc.).

$GLABEL$: This is used to write additional text to the graphics screen. The right argument is the 'user' coordinates of the text, which means XY coordinates in terms of the data you are

plotting, and the left argument is the text desired. *GLABEL* must be executed from within the plotting routine while the graphics display is active. *PLOT* has a comment indicating where *GLABEL* calls can be run. The label will be centered on the coordinates. For labeling in absolute screen coordinates you can use the '⎕*GWRITE*' system function, or the plot background function '`cwrite`', which allows text to be centered.

POINTS: Controls whether data is plotted as continuous lines or as separate points. The default is off (draws lines). The points of each successive data set are marked with a series of symbols. *POINTS* can be used with *SET*, *ON*, and *OFF* similar to the *AUTOSCALE* function i.e.:

> *SET POINTS ON*

NOTE: The argument of *POINTS* can be a vector of 1's and 0's to alternate between drawing points and lines.

PLOTSEQ: Controls the order of linetypes used for plotting. The argument is a string containing any combination of the following linetype names: *DRAW, DOT, SHORTDASH, DOTDASH, LONGDASH*. They can be separated by either spaces or commas.

RESTORE: This is a milder version of *DEFAULT*, in that it only resets the display parameters, without erasing the labels or the video and printer settings. If you want to change the default location or size of the plot on the screen, edit the Viewport setting in *RESTORE*, and run the function once.

SET: This is a 'dummy' function that absorbs the result of some functions that would otherwise clutter up the screen.

WINDOW: This is used to set the plotting scales when autoscaling is off. The syntax is:

> *SET WINDOW   X1 , Y1 , X2 , Y2*
> (lower left)   (upper right)

To check the current setting, run *WINDOW* with an empty vector for an argument (without *SET*) i.e.

> *WINDOW ''*
> 0 0 1 2 9.77E^-4 1.95E^-3   (1st 4 elements = current Window)

Note: For Log plots, the units are the exponents of the coordinates desired. For example, if you want your plot to start at $10E^-6$, you would use '$^-$6' for *WINDOW*.

VIEWPORT: This is used similar to the *WINDOW* function, but it controls the screen coordinates of the plot border. The screen covers an area ranging from 0,0 at the lower left, to 1023,1023 at the upper right. The right argument is XY coordinates of the border corners. The syntax is:

> *SET VIEWPORT   X1 , Y1 , X2 , Y2*
> (lower left)   (upper right)

To check the current setting, run *VIEWPORT* with an empty vector for an argument (without *SET*) i.e.

```
VIEWPORT ''
200 124 1000 924 0.782 0.782   (1st 4 elements = Viewport)
```

Note: For Log plots, the units are the exponents of the coordinates desired. For example, if you want your plot to start at $10E^-6$, you would use '$^-6$' for *VIEWPORT*.

XGRID: This turns X axis grid lines on or off. The default is 'off'. It can be used with the dummy function *SET* and the variables *ON* (= 1) and *OFF* (= 0) i.e. :

```
SET XGRID ON
```

YGRID: This turns Y axis grid lines on or off. The default is 'off'. Operation is similar to *XGRID*.

3.8 HELP FUNCTIONS

The 3 standard help functions described in the introduction are available:

EXPLAIN '*NAME*': Displays a more detailed summary of the function *NAME*

SUMMARY: Lists 1 line descriptions of 'main' functions

SUMMARYALL: Lists 1 line descriptions of all functions

3.9 VARIABLES

This is a brief description of some of the variables used by *PLOT*. Names of variables normally modified by the user are in capital letters, and background variables are in lower-case.

User Variables: The main variables the user will deal with are the labels: *TITLEΔ*, *XΔLABEL*, and *YΔLABEL*

TITLEΔ: This is a string or character matrix that will be placed at the center top of the page. There is room for 3 lines of text. The default value is '*TITLEΔ*'.

XΔLABEL: This is a string or character matrix that will be placed at centered under the plot. There is room for 2 lines of text. The default value is '*XΔLABEL*'.

YΔLABEL: This is a string or character matrix that will be written vertically to the center left of the plot. There is room for 2 columns of text. If the text is formatted as a matrix, each row is assumed to the label for a plot variable, and will be colored to match. The screen can hold roughly 40 characters per column.

ON: This is a dummy variable equal to '1' to be used to set plotting parameters on.

OFF: This is a dummy variable equal to '0' to be used to set plotting parameters off.

Background Variables: These are required by the plotting functions and must be copied with the workspace.

24

vw: This is a 2 x 6 matrix. The 1st row contains the Viewport coordinates and scale factors, and the 2nd row contains the Window coordinates and scale factors.

xytic: This is a 2 x 5 matrix used by the Autoscaling and labeling routines. The 1st row is for X, the 2nd row for Y. The last entry in each row is the approximate number of major tic marks used in autoscaling.

flags: This contains a collection of numbers needed by various functions. There are at least 9 entries:

> *flags*[1] = X axis Log flag (1 = Log)
> *flags*[2] = Y axis Log flag (1 = Log)
> *flags*[3] = Autoscale flag (1 = Autoscale On)
> *flags*[4] = Grid flag (1 = X, 2 = Y, 3 = Both)
> *flags*[5] = Display flag (sets video adapter in 'ginit')
> *flags*[6] = Printer flag (sets printer in 'printplot')
> *flags*[7] = Spare (not used)
> *flags*[8] = Reserved for future use
> *flags*[9-?] = Color numbers for sequential plot lines

lines: This is a matrix containing the linetype codes set by *PLOTSEQ*

pts: A vector of 1's and 0's defining which data sets will be plotted as points (1's) or connected lines (0's). See *POINTS* function.

3.10 SPACE (THE FINAL FRONTIER..)

If you are using *LLPLOT* and are short of memory, you can use the function *PDECOM* located in the *LLUTILTY* workspace to remove the regular comments from the functions. *PDECOM* leaves all of the 'public comments' used by the documentation functions like *EXPLAIN*. If you are still short on room, you can also hardwire your video adapter and printer and strip out the extra code and comments from '*ginit*' and '*print*'.

3.11 CUSTOMIZING

There are a couple of things you may wish to 'play' with. If you don't like the size, placement or shape of the plot, you can change the coordinates called with the *VIEWPORT* function in *RESTORE*. If you want to put additional text on your plot, you can use the *GLABEL* function described above. This must be executed within *PLOT* at the place indicated by a comment. You can also add, subtract, or change the plot colors (stored in 'flags') by altering the vector in *RESTORE*.

3.12 *MARTHA* PLOTTING

This is a description of the special features of the APL plotting workspace that pertain to its use with *MARTHA*. The '*MARTHAP*' workspace can be used for all sorts of general plotting, but when used to display *MARTHA* results, the XY labels are automatically set to match the analysis parameters.

In general, the plotting functions in this workspace should perform exactly like the original line-terminal character plot function in *MARTHA*. It has been tested with a variety of *MARTHA* simulations,

and displays the appropriate labels, curves etc. There are three functions that have special code for *MARTHA* use: *PLOT*, *AND*, and *VS*. All three look for two *MARTHA* variables: '*hd*', which is a character vector containing labeling information for the last analysis executed, and '*of*', which is a numerical vector containing scaling and other plotting data. If either of these do not exist in the workspace, or if *hd* is empty, the plot is drawn using the normal plotting routines and labels, so there is no need to carry around extra variables if the plotting functions are used for other work.

A brief description of the *MARTHA* use of *PLOT*, *AND*, and *VS* follows:

PLOT: This should be fairly transparent. All *MARTHA* requirements should be handled automatically. Because *PLOT* is called by *XLOGPLOT*, *YLOGPLOT* and *XYLOGPLOT*, all of these can be used as well.

AND: This is new to *MARTHA*, and is primarily of use if you have an old result you wish to plot along with a new analysis.   It works the same as for regular plots, except it checks for and removes duplicate frequency data. The Y axis labels won't necessarily match the data, but you can fake this by creating a new *hd*. *hd* must be formatted as a string, with 12 spaces per label, and 24 spaces tacked on the end. The labels are entered from right to left, to match the plotting order of the data.

VS: If used in a *MARTHA* analysis, this behaves identically to the old *VS* in *MARTHA*. For those of you unfamiliar with this, *MARTHA* plots any results against the one variable preceded by *VS* in the argument string. For example:

```
PLOT DB S21 DB S11 VS DB S12 DB S22 OF NET
```

would plot dB S21, dB S11, and dB S22 against dB S12.

There are also two function for plotting Smith Charts; '*SMITH*' and the background function '*cp*'. Running *EXPLAIN* on *SMITH* gives:

```
EXPLAIN 'SMITH'

SMITH Z -- Plots Smith chart from MARTHA impedance data
For example, to plot the reflection coefficient of a 1-port:
    SMITH SC OF Net
```

3.13 ACKNOWLEDGMENTS

Mark Stevens did the bulk of the work on the Log plotting capabilities, and helped debug and polish several other features. Don Boroson originated the approach used to get multiple linetypes and John Kaufmann provided a great deal of useful feedback on features for the latest version. Brian Clifton originated the Smith Chart functions.

4. *LLMARTHA* WORKSPACE

ENHANCED *MARTHA* CIRCUIT ANALYSIS WORKSPACE

4.1 INTRODUCTION

This workspace is a combination of a number of *MARTHA* and *LLAMA* workspaces, and contains the most frequently used functions for general circuit analysis work. The core is the main *MARTHA* workspace, but with all of the plotting functions replaced with the *MARTHAP* versions. It also contains a number of element, wiring and response functions that are frequently needed from the *MARTHA* library workspaces. In addition, there are a number of small utility programs for setting frequency sweeps, calculating Pi and Tee pad values etc. This workspace (assumed to be in Library 8) is copied into many of the other *LLAMA* workspaces by the *COPY △WS* function when they are)LOADed. The objects from *LLMARTHA* are then erased by the *LEAVE* function to avoid cluttering up your disk with multiple copies of *LLMARTHA*. Section 1, which describes the *LLUTILTY* workspace, contains detailed information on how this system works.

Documentation of the *MARTHA* functions can be found in the *MARTHA* manuals, or the *HOWMARTH* workspace. The plotting functions all work with the '*EXPLAIN*' function, as do all of the utility functions. A detailed explanation of the plotting system can be found in Section 3, or in the workspace '*HELPPLOT*', with *MARTHA* specific information in the variable '*HLPMPLOT*'.

Below are some comments and the *EXPLAIN* results for the utility functions that are unique to the *LLMARTHA* workspace:

4.2 FREQUENCY VECTOR FUNCTIONS

MARTHA uses the global variable '*F*' to contain the frequency vector for analyzing circuits. There are three small utility functions that help set up various types of frequency vectors:

```
    EXPLAIN 'LINF'

LINF Fstart(MHz),Fstop(MHz),Nsteps -- Creates a Global Linear Freq Vector 'F'
Creates Nsteps+1 points, so that frequency steps come out even
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.

    EXPLAIN 'LOGF'

LOGF Fstart(MHz),Fstop(MHz),Nsteps -- Creates a Global Log Freq Vector 'F'
Creates Nsteps+1 points, so that frequency steps come out even
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.

    EXPLAIN 'FZEROANDF'

FZEROANDF Fzero(Hz),∆F(Hz),Npts -- Creates a Freq Vector Centered on Fzero
Creates a global 'F' variable over Fzero +/- ∆F/2, with Npts points
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

4.3 ATTENUATOR FUNCTIONS

There are two functions that create *MARTHA* network 'pads':

```
EXPLAIN 'PIPAD'
```

```
Net ← PIPAD Atten(dB) -- Creates a MARTHA network Pi-Pad Attenuator
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

```
EXPLAIN 'TPAD'
```

```
Net ← TPAD Atten(dB) -- Creates a MARTHA network T-Pad Attenuator
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

4.4 CIRCUIT EQUIVALENT FUNCTIONS

There are two functions that are designed to model resonant LC circuits from measured data placed in a *MARTHA FOF* (Function Of Frequency):

```
EXPLAIN 'RPCPOF'
```

```
Out←RPCPOF FOF -- Computes Equiv. Parallel Resistance and Capacitance
Analyzes an impedance FOF (2 col), or 1st parameter in a larger FOF
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

```
EXPLAIN 'TORPXP'
```

```
Out←TORPXP FOF -- Computes Equiv. Parallel Resistance and Reactance
Analyzes an impedance FOF (2 col), or 1st parameter in a larger FOF
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

4.5 MISCELLANEOUS FUNCTIONS

There are two other functions unique to *LLMARTHA*:

```
EXPLAIN 'WILKENSONΔ2'
```

```
Net←WILKENSONΔ2 Fzero(Hz) -- Creates 3-Port Wilkenson Pwr Divider
Designed to be used with nodal wiring functions (see LLMARTHAD WS)
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

```
EXPLAIN 'ZNORM'
```

```
ZNORM Zo -- Sets All MARTHA Impedances = Zo.  If 0=ρZo, Lists Current Values
Sets  ZG ← ZL ← ZN ← ZNIN ← ZNOUT ← Zo
LLAMA Ver. 1.0, 12/31/96, Copyright 1996 by M.I.T.
```

4.6 ACKNOWLEDGMENTS

The original selection of commonly used *MARTHA* functions is from Dave Hodsdon, who also wrote a number of the utility routines.

5. *LLMSDIM* WORKSPACE

'FULL FIELD' MICROSTRIP TRANSMISSION-LINE ANALYSIS

5.1 INTRODUCTION

Microstrip transmission line circuits are very useful, and relatively easy to fabricate. Unfortunately, the propagation of signals on microstrip lines is not easy to analyze, and there are no exact closed-form solutions for even the simplest of geometries. Over the years, a large number of numerical and approximate solutions have been developed to deal with this. One of the most accurate and successful solutions was the MSTRIP FORTRAN program developed by Bryant and Weiss in the late 60's [5.1, 5.2]. This used numerical integration of the field equations to calculate the impedances of single and coupled-pair microstrip lines. This program was used as the foundation of a number of closed-form curve-fit approximations in many RF CAD programs.

In the late 70's, an improved version of MSTRIP [5.3] was modified to be accessible from *APL* running on Lincoln Lab's mainframe. Although slow, this gave users access to very accurate microstrip parameters and allowed the development of microstrip filter synthesis workspaces like *LLCOMBFL* (see Section 6). When the Lincoln mainframe was decommissioned, an intense rescue effort was made to retain this capability. This effort was successful, and a fast accurate version of MSTRIP has been implemented completely in *APL*, and is included in the *LLMSDIM* workspace.

In addition to the *MSTRIP* function, *LLMSDIM* includes three additional functions that are useful for microstrip filter work, as well as a number of background functions required by *MSTRIP*.

5.2 *MSTRIP* FUNCTION

The *EXPLAIN* result for *MSTRIP* is shown below. The function handles both single and coupled-pair microstrip lines. All of the dimensions are normalized to the substrate thickness, and are therefore unitless. The one thing to watch out for is that the cover height is specified FROM THE GROUNDPLANE. This is not the way many programs work, but this convention was used in the original program, and has been maintained for compatibility. *MSTRIP* will check the cover height to make sure that it is either zero (no cover), or ≥ 2. This catches the case where the cover height is equal to the substrate thickness, but is mistakenly referred to the top of the substrate, rather than the ground plane. *MSTRIP* can generate a table of results for different width lines by specifying a '$\Delta W/H1$' (width increment) and 'NT' (number of 'tries'). For a single line (or pair), these are set to zero and 1 respectively. The output includes a copy of the input parameters and then (for a single line) the calculated impedance, velocity, effective dielectric constant and capacitance of the line. For coupled lines, the output parameters are the same, only they are presented in pairs of even- and odd-mode parameters (i.e. Zoe and Zoo). An error is reported when analyzing coupled lines if Zoe < Zoo. This can occur with light coupling if the numerical integration parameters within *MSTRIP* aren't set correctly (see Section 5.4 for details).

```
out←MSTRIP in -- Full-Field Microstrip Line Analysis
Computes the characteristics of single and coupled microstrip lines
This is an APL version of the FORTRAN program MSTRIP described in:
 J.A. Weiss, ADVANCES IN MICROWAVES, Vol. 8, pp. 295-320;
             Academic Press, 1974.
```

The input and output formats match the old mainframe APL MSTRIP to
be compatible with other existing APL functions.

  'in' is a vector, or an N × 5 or 6 matrix. 'in' is dimensionless
  Each row is: W/H1, ΔW/H1, NT, H2/H1, ∈r, [S/H1]
                                                   ↑
    (NT = No. of ΔW/H1 increments)   Coupled Lines Only

 *'out' is always a matrix (although it may have only 1 row)*
  For Coupled Lines: ρout = NT×12
  out= ∈r, H2/H1, S/H1, W/H1, Zoe, Zoo, Ve, Vo, ∈effe, ∈effo, Ce, Co
  For Single Lines: ρout = NT×7
  out= ∈r, H2/H1, W/H1, Zo, V, ∈eff, C
  'out' units are Meters/Sec for V's, and Farads/Meter for C's

TOPOLOGY:

```
                            cover
 _____
                                            ↑
                                            |
   |← W →|←--S--→|← W →|                     H2
    =======       =======          ↓_____   |
 ___                            _____  |
 _____dielectric__∈r_____H1___↓_
        GROUND PLANE                 ↑
```
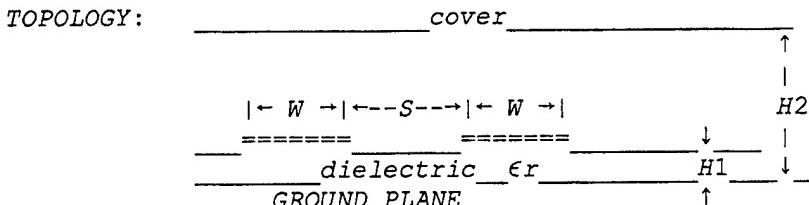
Note! H2 is measured w.r.t. Ground Plane, NOT top of substrate!!

5.3 AUXILIARY FUNCTIONS

    There are three additional user functions in the workspace; '*KLΔMSTRIP*', '*ZFROMKL*' and
'*WSFROMZ*'. *KLΔMSTRIP* is used to calculate the coupling factor '*KL*' for coupled microstrip lines.
This parameter is used in several filter synthesis procedures. *ZFROMKL* is used to compute the odd-
mode impedance of a pair of coupled lines given the even mode impedance and the coupling factor.
WSFROMZ computes the coupled-line width and spacing required to obtain a particular odd- and even-
mode impedance.

out←KLΔMSTRIP params -- Adds the Coupling 'KL' to the Output of MSTRIP
Inputs and outputs are identical to MSTRIP except that
KL = [Zoe-Zoo]÷2×Zoo is catenated onto the end of each output line
'params' is a vector, or an N × 6 matrix
 Each row is: W/H1,ΔW/H1,NT,H2/H1,∈r,S/H1
   (NT = # OF ΔW/H1 Increments)
 'params' is dimensionless
 'out' is also a vector or matrix. ρout = NT×12
out= ∈r,H2/H1,S/H1,W/H1,Zoe,Zoo,Ve,Vo,∈effe,∈effo,Ce,Co,KL

Zoe,Zoo←ZFROMKL Zoe,KL -- Computes Zoo, Given Zoe and KL

out←Ldummy WSFROMZ params -- Computes W/H and S/H from Zoe and Zoo
Synthesizes W/H and S/H for coupled microstrip lines, given Zoe and
Zoo. Uses Newton's Method and MSTRIP to compute line parameters.
Inputs and outputs are compatible with old mainframe WSFROMZ.

SYNTAX: out ← <Ldummy> WSFROMZ H2/H1, ∈r, Zoe, Zoo, <S, W>

```
        L, S, and W are optional 'place-keepers' required
        only to maintain compatibility with Mainframe version

'out' = Last value of:
  €r, H2/H1, S/H1, W/H1, Zoe, Zoo, Ve, Vo, €effe, €effo, Ce, Co, KL

  (Same as MSTRIP output with KL = [Zoe-Zoo]÷2×Zoo catenated on end)

REF: Stanislaw Rosloniec, "Design of Coupled Microstrip Lines By
     Optimization Methods" MTT-35 #11, Nov 1987 pp. 1072-1074
```

5.4 ACCURACY, SPEED, AND MEMORY TRADEOFFS

Inside the $MSTRIP$ program, there are three different places where the computation is 'sub-divided'. The strips are divided into 'M' sub-strips, the Green's function integration is broken up into 'INT' sub-integrals, and the integration itself is performed with an order 'G' Gauss-Legendre Quadrature routine. The number of these sub-divisions has a big effect on the accuracy of the results, but it also affects the speed and memory requirements of the program. Extensive testing has been done to verify the basic operation of the translation, as well as to determine the trade-offs among the three parameters M, INT, and G. The results are described in detail in three sections below.

5.4.1 Accuracy

The first concern was that the results are accurate. In order to verify the performance of the program, a set of 83 test cases was constructed. The first 81 tests include all possible combinations of the following parameters:

$$W/H_1 \quad = 0.1, \ 1, \ 10$$
$$S/H_1 \quad = 0.1, \ 1, \ 10$$
$$H_2/H_1 \quad = 2, \ 11, \ 0 \text{ (no cover)}$$
$$\text{Diel.} \quad = 1, \ 5, \ 10$$

Present curve-fit formulas break down at parameters that can be encountered in the design of Lange Couplers, or with ceramic and super-conducting materials. The following two tests were included to check the performance for extremely narrow strips and at high dielectrics:

$$W/H_1 = 0.01, \ S/H_1 = 0.01, \ \varepsilon_r = 9.9, \text{ No cover}$$
$$W/H_1 = 0.2, \ S/H_1 = 1, \quad \varepsilon_r = 30, \text{ No cover}$$

Internally, $MSTRIP$ starts by computing the strip capacitances and effective dielectric constants, and then derives the impedances and velocities from those. To minimize storage, only the strip capacitances and effective dielectrics were compared.

The first standard for comparison was the battered remains of the old APL/FORTRAN program running on the MIT Campus mainframe. Although difficult to run, it did store valid results in a file before it crashed. When the new program was run with M=20, INT=8, and G=32 the results match to better than 1 part in 10^4 for all 83 cases. These differences are probably due to improvements made in the accuracy of some of the integral routines.

The next step was to increase the number of sub-strips, M, and compare the results against an analytical standard using a technique from Ref. [5.3]. With the cover height set equal to the substrate thickness ($H_2/H_1=2$), microstrip produces identical impedances to a stripline geometry filled with a dielectric of $\varepsilon_{rSL} = \frac{1}{2}(\varepsilon_{rMS} + 1)$. The stripline impedances can be calculated exactly, and Table II from Ref. [5.2] includes a comparison of the impedances calculated by the original MSTRIP program, the improved MSTRIP2 (which is the FORTRAN program we had been running), and the stripline values. The table below (Table 5.1) compares the MSTRIP2 and stripline results from that table to results obtained with the new APL program running with $M=100$, $INT=16$, and $G=32$.

Table 5.1. Computed Impedance Comparison Of Coupled Microstrip Lines

FORTRAN MSTRIP*, *APL MSTRIP***, and Balanced Stripline
H2/H1=2, εr=9.6 for Microstrip, 5.3 for Stripline

S/H1	W/H1	FORTRAN MSTRIP2 Zoe	APL MSTRIP Zoe	STRIP-LINE Zoe	FORTRAN MSTRIP2 Zoo	APL MSTRIP Zoo	STRIP-LINE Zoo
0.5	0.3	89.82	89.46	89.36	58.26	57.87	57.77
	0.8	57.25	56.93	56.84	40.60	40.18	40.07
	1.3	42.46	42.19	42.12	32.43	32.01	31.90
1.0	0.3	81.27	80.91	80.81	67.29	66.92	66.81
	0.8	53.07	52.74	52.65	45.63	45.26	45.16
	1.3	40.07	39.78	39.70	35.60	35.25	35.16
2.0	0.3	75.77	75.41	75.31	72.89	72.52	72.42
	0.8	50.22	49.88	49.78	48.68	48.33	48.23
	1.3	38.39	38.08	38.00	37.47	37.14	37.06
4.0	0.3	74.39	74.03	73.93	74.27	73.91	73.80
	0.8	49.48	49.14	49.05	49.42	49.07	48.98
	1.3	37.95	37.64	37.55	37.91	37.60	37.51
10.0	0.3	74.33	73.97	73.87	74.33	73.97	73.87
	0.8	48.45	49.11	49.01	49.45	49.11	49.01
	1.3	37.93	37.62	37.53	37.93	37.62	37.53

* FORTRAN MSTRIP2 using 20 sub-strips   ** *APL MSTRIP* using 100 sub-strips

For the cases shown, the errors between the new program and the stripline result are typically 0.1 Ohms or less. At $M=20$, the MSTRIP2 program shows errors of 0.6 to 0.3 Ohms. Unfortunately, some internal computations are proportional to M^2, so going to 100 strips from 20 increases the load by a factor of 25, which causes severe memory and speed problems. This is a fairly heavy penalty to pay for a few 10ths of an Ohm!

In order to better understand the tradeoffs among the three parameters; M, INT, and G, a number of experiments were done using the 81 standard tests mentioned above. The baseline was a table of results with $M=60$, $INT=16$, and $G=32$. Percentage errors were calculated for the capacitances and effective dielectrics for comparison. The results are summarized below:

VARYING 'M': M is the most critical parameter, and is most sensitive to tight coupling with wide strips (W/H$_1$=10 and S/H$_1$=0.1) particularly with low dielectrics. Reducing M to 40 introduced differences of ~1.5%, and M=30 increased the differences to ~3%. With M=20 (used by the FORTRAN MSTRIP2 program) errors were as large as 5.2%. An M of 40 is probably a good compromise between accuracy and performance.

VARYING 'INT': Reducing INT from 16 to 4 produced differences of ~1 ppm, when W/H$_1$=S/H$_1$=10. INT=2 increased these errors to ~1%. INT=4 appears to be quite adequate, and cuts the computational load from MSTRIP2 by half.

VARYING 'G': Reducing G from 32 to 20 introduced differences of ~0.1% for the case of W/H$_1$=10, S/H$_1$=10, ε_r=10, H$_2$/H$_1$=11. To be safe, G should probably be left at 32.

5.4.2 Accuracy Enhancements

For a typical range of strip widths, line spacings, dielectrics etc., the best compromise for general use seemed to be M=40, INT=4, and G=32. The combline filter synthesis routines in the $LLCOMBFL$ workspace call $MSTRIP$ repeatedly, and G was set to 20 in that version to get slightly faster response.

After the analysis above had been completed, and while developing some narrow band filter designs, a problem was discovered in $MSTRIP$'s results. The narrow bandwidths require lightly coupled lines, and $MSTRIP$ was predicting even-mode impedances lower than the odd-mode impedance, which is physically impossible. This occurred at S/H values above roughly 20, but it was unclear how inaccurate the results might be at narrower spacings. The original tests only evaluated $MSTRIP$ up to an S/H of 10, where there was no indication of trouble.

The spacing at which Zoe first becomes less than Zoo is largely independent of the other microstrip dimensions and ε_r, and grows roughly linearly as the integration parameter INT is increased. Going back to INT=8 (which was used in the original FORTRAN program) moved the problem out to S/H > 40. Rather than pay a computational penalty by fixing INT to a large value, INT can be increased as a function of S/H. The new version of APL $MSTRIP$ uses INT=4 up to S/H=16, and increases INT by 1 for every increase of 4 in S/H beyond that. For most applications, there will be no speed penalty at all. A quick test of the $LLCOMBFL$ filter workspace showed that using G=20 aggravates the problem, so G was set back to 32 there as well.

Figure 5.1 shows a comparison of the even and odd-mode impedances calculated with INT=4, and using the new INT=f(S/H). When INT=4 and the problem manifests itself, the errors grow wildly as S/H increases. However, the spacings required to get serious errors aren't likely to be encountered except in extreme cases. When the problem was discovered, fixing $MSTRIP$ was the only way to find out how large the errors might be, because none of the curve-fit microstrip solutions are valid at these spacings.

EFFECT OF INTEGRATION PARAMETER 'INT' ON MSTRIP
COUPLED LINE IMPEDANCES (W/H=1, er=10, NO COVER)
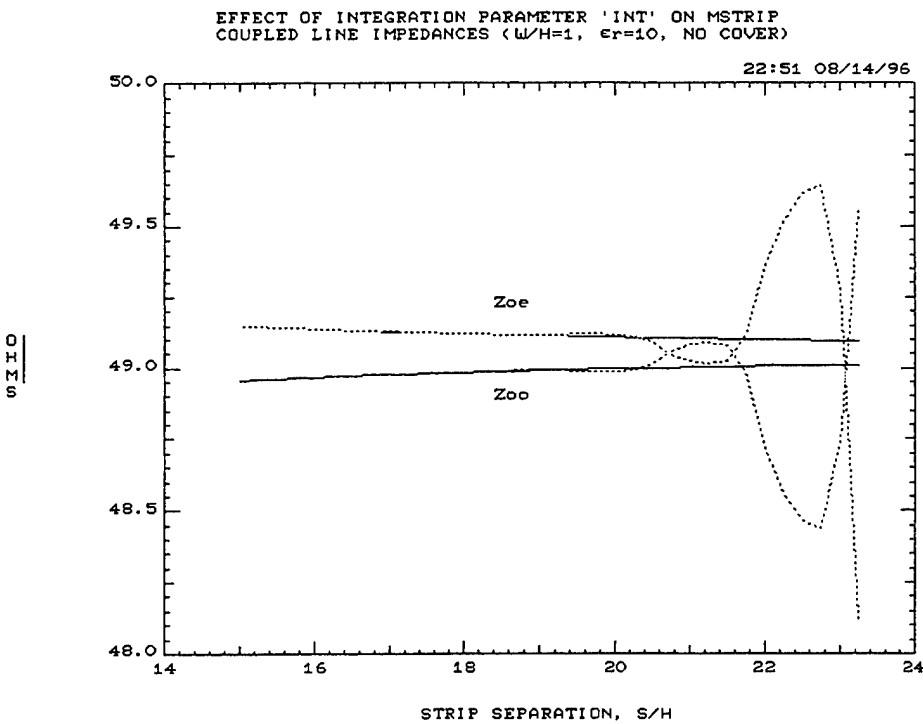
Figure 5.1. Coupled microstrip impedance problem (new version ———, old version ......)

5.4.3 Speed

Because APL is an interpreted language, program loops require re-interpretation of the code each time through the loop. FORTRAN makes extensive use of loops, because it lacks the array operations available in APL. The original translation of MSTRIP into APL retained all but the simplest loops, and required almost a full minute to analyze a single pair of coupled lines (using $M=20$, $INT=8$, $G=10$) on a 33 MHz '486 PC. The integration routine at the heart of the Green's function analysis was being called (and re-interpreted) 4,720 times. By converting the integration to a three dimensional array computation, it gets called twice. Similar changes were made in other parts of the code, and the execution time was reduced to 1.48 seconds!

Unfortunately, the accuracy issues described above indicated that $M=40$, $INT=4$, and $G=32$ may be a better parameter set to work with, and this boosted the execution time back up to 5.7 seconds. This is probably comparable to what the 1978 mainframe took to produce less accurate results. Depending on what sort of materials and dimensions you are working with, it may be possible to get perfectly adequate results using $M=20$ and $G=20$, which should run in under 30 seconds on a '286 machine. Changing M is a simple 1 line edit in the main $MSTRIP$ function. Altering G requires a single change in the '$glqint$' function, but it also requires a background variable containing the appropriate weights and abscissas. The variables for $G=32$ and $G=20$ are included in the $LLMSDIM$ workspace.

The last major change affecting speed was actually necessitated by memory limitations (see below). In order to reduce the memory requirements during the Green's function integration, the loop over INT sub-divisions was placed back into the '$mgreen$' function. This loop is only executed 4 times now (for all but very light coupling), so the overhead of re-interpreting each pass is minimal. This increased the

34

execution time by less than 5%. For people working with M=20, the function $mgreen$ is clearly commented so that this loop can be eliminated again, if desired.

5.4.4 Speed Enhancements

Since the work above was completed, some additional speed improvements have been made. The original FORTRAN program included a fairly direct coding of the Green's function for microstrip lines. Doug Dugas developed several simplifications based on trigonometric identities that reduce the complexity of the computations significantly. Applying these changes to the APL '$mgreen$' function resulted in a 17% speed improvement. Personal computer technology has also improved. On a 100 MHz Pentium system, $MSTRIP$ runs in 1.7 seconds under $APL+PC$ and in 0.44 seconds under $APL+DOS$ (which uses 32 bit operations).

5.4.5 Memory

With the major computational loops removed from the APL code, there are two places where the memory requirements can produce a $WS \ FULL$ message. The first is in the Green's function integration routines, and the second involves matrix inversions. APL uses 64 bit floating point numbers, so it uses up memory very quickly on large arrays.

The Green's function integration is performed using an M by INT by G/2 array. For M=40, INT=4, and G=32, this requires working with an array of 2,560 floating point numbers (~20 Kbytes), including operations with background arrays of similar size. This will execute in APL2000's $APL+PC$ standard system, with DOS 5.0, but requires an efficient memory manager to free up as much of DOS's 640K memory as possible. There is no way it could execute within a $MARTHA$ application. Eliminating the INT dimension from the array reduces the space requirements by a factor of 4, at only a minor impact on speed (see above).

The second memory bottleneck involves matrix inversion. After the Green's function is integrated, an M by M matrix is formed, which must be inverted to compute the charge distribution on the strips. Although APL has an exceedingly efficient matrix inversion operator, it has its limits. M=40 works fine, but because the amount of computation tends to grow as M^2, larger M's could be a problem. M=60 appears to be about the largest array that can be handled in APL2000's $APL+PC$. This should provide more than adequate accuracy for all but the most demanding jobs. The newer APL interpreters available ($APL+DOS$ and $APL+WIN$) can use high memory, which eliminates the problem entirely.

5.5 ACKNOWLEDGMENTS

Doug Dugas did a lot of the archeological work required to get the FORTRAN source code for MSTRIP2 and helped cross-check the APL version against his C version. Doug developed the simplified equations to speed up the $mgreen$ function, and was the first to suggest the INT parameter as the likely source of the wide spacing problem. Jerry Weiss was also very helpful in providing some historical insight and references on the original program. The three auxiliary functions were originally developed on the Lincoln mainframe computer by Dave Hodsdon.

5.6 REFERENCES

5.1 Bryant, T.G., and Weiss, J.A., "Parameters of Microstrip Transmission Lines And of Coupled Pairs of Microstrip Lines," IEEE Trans. Microwave Theory & Techniques, Vol. 16, 1968, pp. 1021-1027

5.2 J.A. Weiss, "Microwave Propagation on Coupled Pairs of Microstrip Transmission Lines," ADVANCES IN MICROWAVES, Vol. 8 Academic Press, New York, 1974, pp. 295-320

5.3 J.A. Weiss, R.S. Withers & R.C. Lewis, "MSTRIP2: Parameters of Microstrip Transmission Lines and of Coupled Pairs of Lines -- 1978 Version and Its Application," Lincoln Laboratory Technical Report No. 600, 4 June 1982

6. *LLCOMBFL* WORKSPACE

MICROSTRIP COMBLINE FILTER SYNTHESIS

6.1 INTRODUCTION

This workspace contains functions to synthesize tap-fed microstrip combline filters. The synthesis starts by executing the '*HELP*' function, which prompts the user for the required inputs. Following the completion of a given design, it may be analyzed using any available '*MARTHA*' functions. A display of all of the filter parameters is produced, and they are stored for editing later.

In order to have all the *MARTHA* functions available to work with the results, loading *LLCOMBFL* automatically copies the *LLMARTHA* workspace, which is assumed to be in Library 8. If you are using a different library for the *LLAMA* workspaces, you with need to edit the argument of the *COPY△WS* command in the *START* function as described in Sections 1.8 and 2.2.

6.2 TOPOLOGY

The combline filter consists of a number of electrically-short ($<< \frac{1}{4} \lambda$) parallel coupled microstrip lines. The layout of a combline filter is shown in Figure 6.1. The lines are grounded on one end, and resonated by capacitors (usually variable) on the far end. The line and substrate parameters are specified to be compatible with the MSTRIP analysis program.
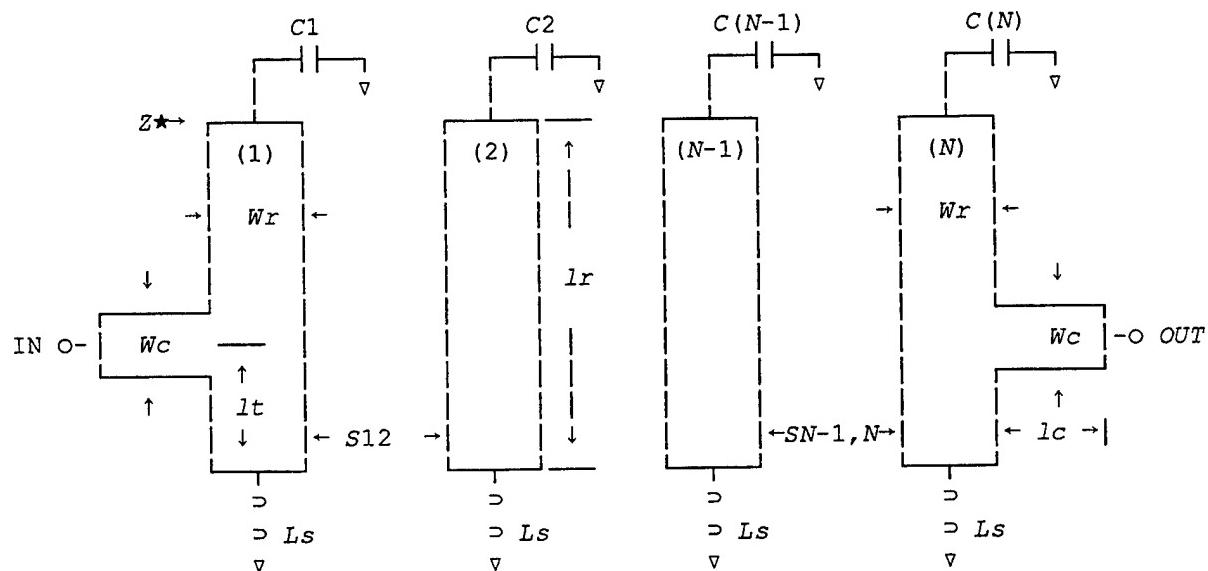
NOTE: All resonators are the same length and width

Figure 6.1. Layout of combline filter

Figure 6.2 shows how the line-spacing and widths are dimensioned, along with the substrate thickness and cover-height. Cover-height is measured relative to Ground Plane, not substrate!

```
_____COVER_____
                                              ↑
                                              |
      |←Wr→|← S12 →|←Wr→|← S23 →|←Wr→|        H2    NOTE! H2/H1=0
      ======        ======       ======              puts cover at
_____        _____       _____  |       Infinity
                                              ↑  |
_____dielectric__€r_____H1 |_
                                              ↓_____↓_
_____
                    GROUND PLANE
```
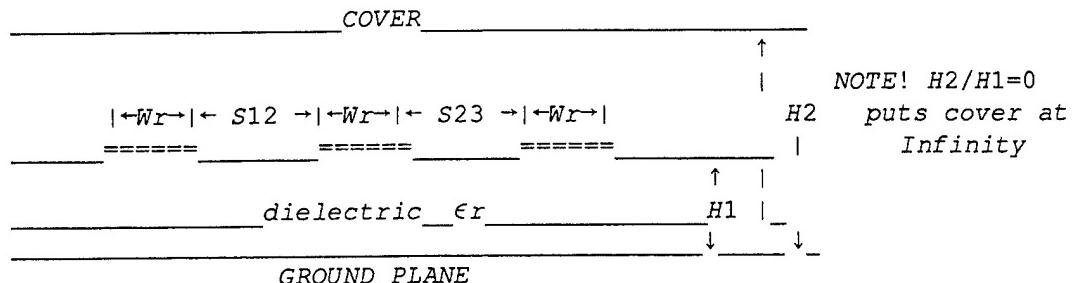
Figure 6.2. Cross-section of combline filter substrate & cover

6.3 PARAMETER INPUT

'*HELP*' prompts the user for the following 14 inputs:

1. In-Band Ripple (dB): 0 = Butterworth Response

    (>0) = Chebyshev Response

    ⁻1 = Constant K Response

    ⁻2 = Gaussian Amplitude Response

2. Number of Poles (≥ 2)

3. Center Frequency in MHz

4. 3 dB (!) Bandwidth in MHz (Regardless of Filter type)

5. Source Resistance in Ohms

6. Resonator Line Width in Inches (Wr)

7. Resonator Electrical Length in Degrees

8. Resonator 'Short' Inductance in NanoHenries

9. Substrate Relative Dielectric Constant (ϵr)

10. Substrate Thickness in Inches ($H1$)

11. Cover Height in Inches ($H2$)

12. Input Coupling Line Width in Inches (Wc)

13. Input Coupling Line Length in Inches (lc)

14. Filter Name: Any valid APL variable name. This becomes a Global variable. After the analysis, this variable will contain a *MARTHA* compatible network description of the filter.

6.4 COMBLINE FILTER EXAMPLE

Below is an example listing of a session used to create a 2nd order Butterworth filter:

```
     HELP
N POLE MICROSTRIP COMBLINE FILTER SYNTHESIS

DO YOU WISH TO MODIFY AN OLD DESIGN (Y OR N)? N
ENTER RIPPLE(DB), POLES, CENTER FREQ(MHZ), BW(MHZ,3 DB), RGEN(OHMS)
□:
     0 2 1000 100 50
ENTER RESONATOR WIDTH(INCHES), LENGTH(DEG), SHORT INDUCTANCE(NANOHENRIES)
□:
     0.1 40 0.2
ENTER SUBSTRATE DIELECTRIC(ER), THICKNESS(H1,INCHES), COVER
HEIGHT(H2,INCHES)
□:
     2.56 0.06 1
ENTER COUPLING LINE WIDTH(WT,INCHES), LENGTH(LT,INCHES)
□:
     0.1 0.5
WHAT IS FLT NETWORK NAME TO BE? DEMO
```

NO	PARAMETER	VALUE	UNITS
1	RIPPLE	.000	DB
2	POLES	2.000	
3	CENTER FREQUENCY	1000.000	MHZ
4	BANDWIDTH, 3DB	100.000	MHZ
5	SOURCE RESISTANCE	50.000	OHMS
6	RESONATOR WIDTH	.100	INCHES
7	RESONATOR LENGTH	40.000	DEG
8	SHORT INDUCTANCE	.200	NANOHENRIES
9	SUBSTRATE DIELECTRIC	2.560	
10	SUBSTRATE THICKNESS	.060	INCHES
11	COVER HEIGHT	1.000	INCHES
12	INPUT LINE WIDTH	.100	INCHES
13	INPUT LINE LENGTH	.500	INCHES
14	FILTER NAME	DEMO	

```
ENTER PARAMETER NO. TO BE CHANGED (0, IF OK): 0   (←NOTE! You can edit your inputs
SYNTHESIS STARTED AT 14:59 HOURS, FEB 07, 1992        before beginning synthesis)
```

6.5 SAMPLE COMBLINE SYNTHESIS OUTPUT

```
8 LLMARTHA PCOPIED                    (← ** SEE NOTE BELOW **)
MICROSTRIP COMBLINE FILTER 'DEMO
14:59 HOURS, FEB 07, 1992


CENTER FREQUENCY          = 1000.000   MHZ
3 DB BANDWIDTH            =  100.000   MHZ
IN-BAND RIPPLE           =     .000   DB
NUMBER OF POLES          =    2.000
SOURCE RESISTANCE        =   50.000   OHMS
FILTER RESISTANCE        =  621.880   OHMS
SHORT INDUCTANCE         =     .200   NANOHENRIES


RESONATOR IMPEDANCE      =   73.347   OHMS
RESONATOR CAPACITANCE    =    2.875   PF          (1)
RESONATOR CAPACITANCE    =    2.875   PF          (2)


RESONATOR LINE WIDTH     =     .100   INCHES
RESONATOR LINE LENGTH    =     .897   INCHES      (40 DEG)
RESONATOR SPACING        =     .123   INCHES      (1-2)


COUPLING LINE IMPEDANCE  =   69.076   OHMS
COUPLING LINE WIDTH      =     .100   INCHES
COUPLING LINE LENGTH     =     .500   INCHES
COUPLING LINE TAP        =     .249   INCHES


SUBSTRATE DIELECTRIC     =    2.560
SUBSTRATE THICKNESS      =     .060   INCHES
COVER HEIGHT             =    1.000   INCHES

ELEMENT MATRIX:

2.00E⁻10 .022775 .006322 .012700   69.08 2.09E08 2.053  1.667 2.87E⁻12
622


 73.35   64.66  2.05E08  2.14E08  2.134   1.962 1.667  2.050  2.87E⁻12
.07E⁻2

    RUN TIME IS   .68 MINUTES
```

** NOTE: In order to run, the functions 'getcr' and 'gettap' may need editing to contain the
correct library number of your LLMARTHA workspace!

6.6 ELEMENT MATRIX KEY

1st Row Elements: 1) Short Inductance (Henries)

2) Resonator Length (Meters)

3) Tap Position (Meters)

4) Input Line Length (Meters)

5) Input Line Zo (Ohms)

6) Input line Phase Velocity (Meters/Sec)

7) Input Line Effective Dielectric Constant

8) Input Line Width, $Wc/H1$

9) 1st Resonator Capacitor, $C1$ (Farads)

10) Impedance at Resonator End ($Z\star$ on topology diagram)

Nth Row Elements   1) Even Mode Impedance (Line N-1 to N) (Ohms)

2) Odd Mode Impedance (Line N-1 to N) (Ohms)

3) Even Mode Velocity (Line N-1 to N) (Meters/Second)

4) Odd Mode Velocity (Line N-1 to N) (Meters/Second)

5) Even Mode Effective Dielectric (Line N-1 to N)

6) Odd Mode Effective Dielectric (Line N-1 to N)

7) Resonator Width, $Wr/H1$

8) Resonator Spacing, $(S(N-1),N)/H1$

9) Resonator Capacitor, $C(N)$ (Farads)

10) Coupling Coefficient, $K(N-1),N$

6.7 ACKNOWLEDGMENTS

This workspace was originally developed on the Lincoln Lab mainframe by Dave Hodsdon, using the old FORTRAN version of MSTRIP (see the $LLMSDIM$ section for more details on this).
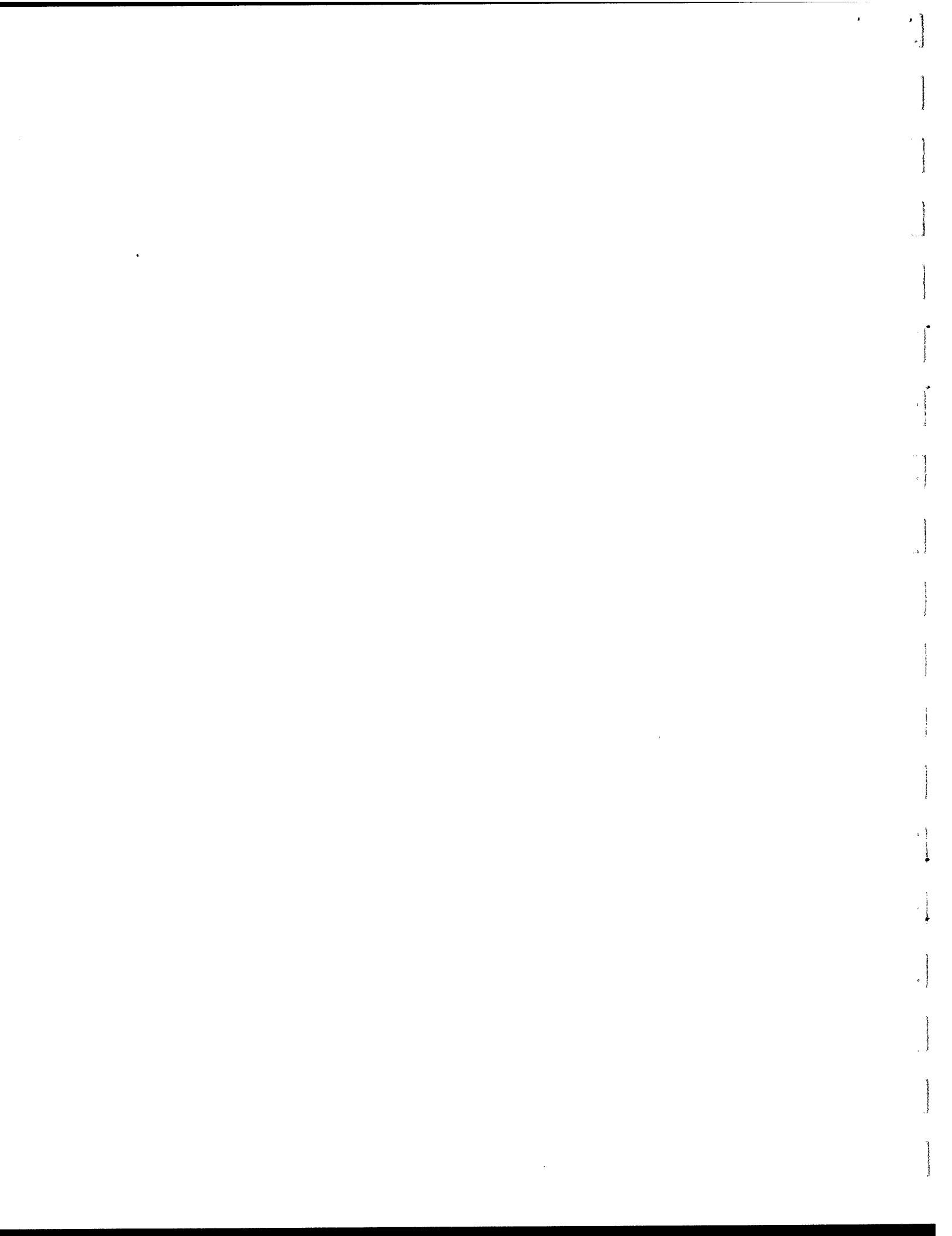
7. *LLFILTER* WORKSPACE

FILTER DESIGN WORKSPACE

7.1 INTRODUCTION

This workspace contains a variety of functions for synthesizing and modifying passive and active filters. All of the functions have been set up to work with the *EXPLAIN* function, and most are designed to prompt the user for input if an empty vector is given as a right argument. For example, executing

```
    LPGAUSS ''
```

produces the following prompt for input:

```
 Npoles,Fcutoff(Hz),Zo(Ohms):
```

The functions in this workspace have been grouped into eleven different categories, and are described in detail below. In addition to calculating fairly standard LC filters, there are a number of functions to aid in designing practical filters with realizable component values using impedance transformations and the like. Many of the functions allow the inclusion of parasitics, and produce *MARTHA* network descriptions that can be analyzed, plotted or combined with other circuits. Because this workspace makes extensive use of *MARTHA*, loading *LLFILTER* copies the *LLMARTHA* workspace (assumed to be in Library 8). If you are not using library 8, you with need to edit the argument of the *COPY∆WS* command in the *STARt* function as described in Sections 1.8 and 2.2.

7.2 BASIC BAND-PASS FILTER FUNCTIONS

The functions '*BPFILTER*' and '*BPGAUSS*' implement standard LC band-pass filters.

```
net←'LorC' BPFILTER params -- Synthesizes LC Band-Pass Filters
Uses 'proto' fn to synthesize Tchebysheff or Butterworth filters.
'LorC' = 'L' or 'C', indicates end element type.
'params' is a five element vector:
    params[1] = In-Band Ripple (dB) (Zero for Butterworth)
    params[2] = Number of Poles
    params[3] = Center Frequency (Hz)
    params[4] = Bandwidth (Hz) (Butterworth→3 dB, Tchebysheff→Ripple)
    params[5] = Generator Impedance
'net' = MARTHA network description with ZG and ZL specified
Prompts for input if 'LorC' missing or invalid, or if ρparams≠5
```

```
net←BPGAUSS Np,Fo(Hz),BW(Hz),Zo(Ohms) -- Synthesizes Gaussian BP Filters
Performs a band-pass transformation on the results from 'LPGAUSS'.
Bandwidth is 3 dB.  Prompts for parameters if any inputs are missing.
2≤Npoles≤11
```

7.3 'HIGH SIDE C' CAPACITIVELY COUPLED BAND-PASS FILTER FUNCTIONS

The main function in this group is '*HSCBPF*', which synthesizes a band-pass filter consisting of resonators coupled with capacitors. This type of filter is suitable for narrow band designs where reduced

attenuation at high frequencies is not a problem. The function '*HSCBPFNET*' is used to convert the output of *HSCBPF* into a *MARTHA* network. Although easy and economical to build, the component values and impedance of the resulting filter design can be awkward. The functions '*ENDCAP*', '*LTRANS*' and '*TEECIJ*' can be used to modify the design to transform the impedance and correct for component value problems.

```
Z←HSCBPF params -- Synthesizes a High-Side C Coupled Band-Pass Filter
Uses 'k' and 'q' values from 'kqvalues', with the resonator inductance
forced.  Typically used for fractional BW's less than 10 percent, and
where finite high-frequency attenuation is acceptable.
'params' is a 5 element vector:
    params[1] = In-Band Ripple (dB)  (0→Butterworth,  (>0)→Tchebycheff
                                      ⁻1→Constant K,  ⁻2→Gaussian)
    params[2] = Number of Poles
    params[3] = Center Frequency (Hz)
    params[4] = Bandwidth (Hz) (Ripple for Tchebysheff, 3 dB for others)
    params[5] = Resonator Inductance (Henries)
Prompts for input if parameters missing, or if number of params≠5
The output vector 'Z' is equal to the input vector 'params' with the
following items catenated:
    Z[6] = Load Resistance (Ohms)
    Z[7 8] = 0, or L[1] & L[N] Resonator Inductors For Gaussian Filter
    Z[9 to 11] = 0 (Reserved for use by 'TEECIJ' and 'ENDCAP')
    Z[12 to nn)] = (Np) Resonator Capacitors (Farads)
    Z[(nn+1) to mm] = (Np-1) Coupling Capacitors (Farads)
When the resonator inductance is pre-determined, the filter impedance
is typically NOT 50 Ohms.  The function ENDCAP can add a capacitive
impedance divider to the output of HSCBPF to correct this.  If the
coupling capacitor values are too small, the function TEECIJ can be
used to increase their value at the cost of additional capacitors.
The function HSCBPFNET is used to convert the result of any of the
above functions into a MARTHA network description.
Notes: Constant K design produces minimum loss for a given stop-band.
Gaussian filters are inherently asymmetric, and only the inner L's
can be set equal to params[5].  The 1st & last inductor are set so
that the inner inductor(s) = geometric mean of L[1] & L[N].


net←Q HSCBPFNET X -- Converts Output From 'HSCBPF' Into MARTHA Network
Creates a MARTHA network description of a high-side C-coupled band-pass
filter synthesized using 'HSCBPF'.  'X' is the output from 'HSCBPF',
with or without modifications from 'ENDCAP' or 'TEECIJ'.  'Q' is the
coil Q at Fo.  'Q'=0 assumes infinite coil Q.  If 'Q' is omitted, fn
prompts for input.


V←Ro ENDCAP X -- Adds Capacitive Imp. Transformer To Output Of HSCBPF
Converts the input/output resonator capacitances from 'HSCBPF' to a
capacitive divider to transform the resonator impedance (Rp) down to a
desired input/output impedance (Ro).  'X' is the output from 'HSCBPF'
or 'TEECIJ'.  The output vector 'V' = 'X', but with:
V[6]=Ro,V[10]=Ctop,V[11]=Cbot,V[14 to nn]=Cr's,V[(nn+1) to mm]=Cij's.
NOTE: The capacitance of the matching network is subtracted from the
end resonator capacitances.  If the impedance transform required is
too small, this can result in negative resonator capacitors!  If this
occurs, you can find the impedance transform where the Cr's=0.  Then
```

transform the remaining amount using standard impedance scaling.

V←Cs TEECIJ X -- Modifies High-Side C Coupling to Capacitor 'Tee's
This can be used to raise the values of the coupling-network capacitors
in a high-side C coupled filter. 'X' is the output of either 'HSCBPF'
or 'ENDCAP'. 'Cs' is the desired shunt capacitance. If 'Cs' is
omitted, fn prompts for input. The output 'V' is vector equal to 'X',
but with V[9]=Cs, and modified Cr's, and Cij's

els←L' LTRANS L,C0,C1,C2 -- Imp. Transform for Cap-Coupled Shunt Tanks
Performs transform in Geffe, 'Simplified Modern Filter Design', p34,
Fig 4-8. Typically used to increase size of tank inductors.

```
    C1      C2                      C1÷N        C2÷N
  o-||--o--||--o             o--o-----||---o---||-----o--o
       |                          |            |              |
     |--o--|                      |         |--o--|           |
 C0  |     ⊏     ---\      C1×α |   C3 |    ⊏ L×   | C2×α
  ===      ⊏ L   ---/       ===     ===    ⊏ N★2   ===
  |        ⊏                      |         |      ⊏      |
  |--o--|                         |      |--o--|          |
     ▽                            ▽        ▽              ▽
```
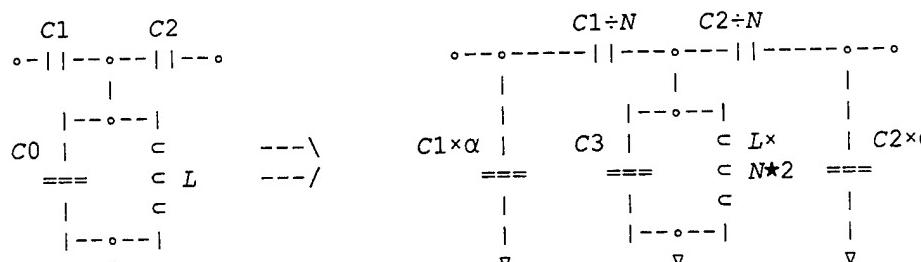
$$\alpha = (N-1) \div N \qquad C3 = (C0 - (N-1) \times (C1+C2)) \div N \star 2$$

L' = Desired value for (L×N★2)
els = (L×N★2), (C1×(N-1)÷N), (C1÷N), C3, (C2÷N), C2×(N-1)÷N

7.4 'HIGH SIDE L' INDUCTIVELY COUPLED BAND-PASS FILTER FUNCTIONS

High-Side-L filters are similar to the High-Side-C design, only the resonator sections are coupled with inductors. This type of filter has poor attenuation at low frequencies, but rolls off more at higher frequencies. In general, the response of inductively coupled filters will be more symmetrical than capacitively coupled designs. The function 'HSLBPF' does a straight forward synthesis, where the 'HSLCBPF' function includes the effects of finite Q and parasitic capacitance of the coupling inductors. Both functions return a *MARTHA* compatible network for further analysis.

net←HSLBPF params -- Synthesizes High-Side L Coupled Band-Pass Filters
Creates a MARTHA network description with ZG and ZL set to Rsource.
Computes coupling using K and Q values from 'kqvalues' fn.
'params' is a five element vector:
*   params[1] = In-Band Ripple (dB) (0→Butterworth, (>0)→Tchebycheff*
*   ⁻1→Constant K, ⁻2→Gaussian)*
*   params[2] = Number of Poles (≥2)*
*   params[3] = Center Frequency (Hz)*
*   params[4] = Bandwidth (Hz) (Ripple for Tchebysheff, 3 dB for others)*
*   params[5] = Load and Generator Resistance (Ohms)*
Prompts for input if any parameters are missing. Typically useful for
10 percent fractional BW or less. See 'HSLCBPF' fn to account for
coupling inductor parasitics and finite Q.
Notes: Except for Gaussian designs, resonator inductors are identical.
Gaussian filters are inherently asymmetric, and only the inner L's
can be set equal. The 1st & last inductor are set so that the inner

```
inductor(s) = geometric mean of L[1] & L[N].
Constant K design produces minimum loss for a given stop-band.


net←Q HSLCBPF params -- High-Side L BP Filter Synthesis w/Parasitics
Creates a MARTHA network for a high-side L coupled band-pass filter
with finite Q and coupling inductor parasitic capacitance included.
'params' is a 7 element vector:
  params[1] = In-Band Ripple (dB) (0→Butterworth, (>0)→Tchebycheff
                                    ‾1→Constant K, ‾2→Gaussian)
  params[2] = Number of Poles
  params[3] = Center Frequency (Hz)
  params[4] = Bandwidth (Hz) (Ripple for Tchebysheff, 3 dB for others)
  params[5] = Source Resistance (Ohms)
  params[6] = Resonator (Shunt) Inductance (Henries)
  params[7] = Coupling Inductor Parasitic Capacitance (Farads)
Prompts for input if any parameters or Q are missing.
Function sets: ZG←ZL←ZN←ZNIN←ZNOUT←Rsource
'Q' is coil Q: Q=0 denotes infinite Q, function sets Q=1E12
               Q>0 adds series R to coils, Q=2×PI×Fo×Lo÷Rseries
Notes: This function synthesizes an LC ladder filter, and then adds
a series-L, shunt-C, series-C matching network.  Because the
impedance transform is added to an existing network, it includes a
negative capacitor to cancel the shunt-C at the filter input.  If
the impedance transformation required is too low, the matching
network may have negative elements, requiring a large value for L.
Except for Gaussian designs, the resonator inductors are identical.
Gaussian filters are inherently asymmetric, and only the inner
inductors can be set to the specified value.  The 1st & last inductor
are set so the desired value (params[6]) = geometric mean of L[1] & L[N].
Constant K design produces minimum loss for a given stop-band.
```

7.5 IMPEDANCE TRANSFORMING BAND-PASS FILTERS

The function 'BPXFMR' synthesizes a Tchebysheff band-pass filter with unequal input and output impedances. This is frequently used more for its impedance matching characteristics than as a filter, but it is often handy to be able to combine the two functions in one network. The function currently only does second order filters, but there are references that could be used to expand its capabilities.

```
net←BPXFMR params -- Synthesizes a BP Filter Impedance Transformer
Synthesizes a Tchebyscheff bandpass-filter impedance transformer using
the background fn 'protog'.  'params' is a seven element vector:
    params[1] = In-Band Ripple (dB) (0 Not Allowed)
    params[2] = Number of Poles (Only N=2 Implemented)
    params[3] = Center Frequency, Fo (Hz)
    params[4] = Ripple Bandwidth, BW (Hz)
    params[5] = Generator Resistance, ZG (Ohms)
    params[6] = Load Resistance, ZL (Ohms)
    params[7] = Option Selection (1 - 5)
Selectable Options Are:
    1 = Inductive 'PI' Reactive Transformer
    2 = Inductive 'TEE'    ''            ''
    3 = Capacitive 'PI'    ''            ''
    4 = Capacitive 'TEE'   ''            ''
```

```
        5 = Maximum Transformer Turns Ratio
```

'net' is a MARTHA network description with ZG and ZL specified, ZG≤ZL.
Prompts for inputs if any parameters are missing.

The synthesis procedure is: 1) design low-pass prototype, 2) bandpass
transform to a center freq. of (Fo÷BW), 3) insert the selected
reactance transformer, 4) frequency and impedance scale to the desired
center frequency and generator resistance.

References:
1. Geffe,Philip R.,'Simplified Modern Filter Design,' Hayden Book
     Company,Inc.,New York, 1966 pp.33-35
2. Plotkin,S. AND Nahi,N.E., 'On Limitations of Broad-Band Impedance
     Matching Without Transformers', IRE Transactions on
     Circuit Theory, June 1962, pp.125-132

7.6 BAND-STOP FILTER FUNCTION

The function 'BSFILTER' creates a MARTHA network description of a simple LC band-stop filter. You can specify whether the end element of the filter is an inductor or a capacitor.

```
net←'LorC' BSFILTER params -- Synthesizes LC Band-Stop Filters
Uses 'proto' fn to synthesize Tchebysheff or Butterworth filters.
'LorC' = 'L' or 'C', Indicates end element type.
'params' is a five element vector:
    params[1] = In-Band Ripple (dB) (Zero for Butterworth)
    params[2] = Number of Poles
    params[3] = Center Frequency (Hz)
    params[4] = Bandwidth (Hz) (Butterworth→3 dB, Tchebysheff→Ripple)
    params[5] = Generator Impedance
'net' = MARTHA network description with ZG and ZL specified
Prompts for input if 'LorC' missing or invalid, or if ρparams≠5
```


7.7 HIGH-PASS FILTER FUNCTION

'HPFILTER' creates a MARTHA network description of a simple LC high-pass filter. You can specify whether the end element of the filter is an inductor or a capacitor.

```
net←'LorC' HPFILTER params -- Synthesizes LC High-Pass Filters
Uses 'proto' fn to synthesize Tchebysheff or Butterworth filters.
'LorC' = 'L' or 'C', Indicates end element type.
'params' is a four element vector:
  params[1] = In-Band Ripple (dB) (Zero for Butterworth)
  params[2] = Number of Poles
  params[3] = Cut-Off Freq (Hz) (Butterworth→3 dB, Tchebysheff→Ripple)
  params[4] = Generator Impedance
'net' = MARTHA network description with ZG and ZL specified
Prompts for input if 'LorC' missing or invalid, or if ρparams≠4
```

7.8 LOW-PASS FILTER FUNCTIONS

There are two functions for synthesizing low-pass filters, '*LPFILTER*' and '*LPGAUSS*'. *LPFILTER* will do either Butterworth or Tchebysheff designs, and *LPGAUSS* is used for Gaussian response filters. Both functions return a *MARTHA* network description.

```
net←'LorC' LPFILTER params -- Synthesizes LC Low-Pass Filters
Uses 'proto' fn to synthesize Tchebysheff or Butterworth filters.
'LorC' = 'L' or 'C', Indicates end element type.
'params' is a four element vector:
  params[1] = In-Band Ripple (dB) (Zero for Butterworth)
  params[2] = Number of Poles
  params[3] = Cut-Off Freq (Hz) (Butterworth←3 dB, Tchebysheff←Ripple)
  params[4] = Generator Impedance
'net' = MARTHA network description with ZG and ZL specified
Prompts for input if 'LorC' missing or invalid, or if ρparams≠4


net←LPGAUSS Np,Fc(Hz),Zo(Ohms) -- Synthesizes Low-Pass Gaussian Filters
Uses 'lpΔgauss' fn to synthesize LC ladder low-pass Gaussian filters.
'net' is a MARTHA network description.  Prompts for parameters if
inputs missing or wrong number.  2≤Npoles≤10
```

7.9 RESONATOR MODELING FUNCTIONS

There are two functions in the *LLFILTER* workspace that create *MARTHA* network models of crystal or SAW (Surface Acoustic Wave) resonators.

```
net←CRYSTAL Fs,Rs,Q,Ccase -- Creates MARTHA Network Model of a Crystal

net←SAWRESONATOR params -- Creates a MARTHA Model of a SAW Resonator
'params' is a six element vector:
  params[1] = Series Resonant Freq, Frs (Hz)
  params[2] = Unloaded Q
  params[3] = Series Resistance, Rs (Ohms)
  params[4] = Output Capacitance, Co (Farads)
  params[5] = Package Capacitance, Cp (Farads)
  params[6] = Phase (+1=0 Deg,¯1=180 Deg)
'net' = MARTHA network description of the following circuit:


                       Cp
       ------------||--------------
      |            Ls   Cs   Rs   |
IN o--o-----   ---nnn---||---^^^---o--o OUT
      |    ⊃ ⊂                     |
Co === ⊃ ⊂ ← Xfmr for         === Co
      |    ⊃ ⊂   phase             |
      ∇    ∇ ∇                      ∇


Prompts for input if no. of params ≠ 6
```
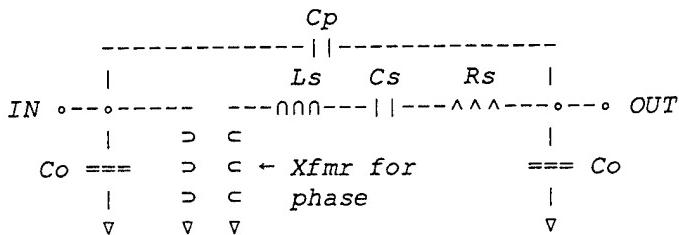
7.10 NORMALIZED STOP-BAND FREQUENCY FUNCTIONS

The functions '*BOMEGAS*' and '*COMEGAS*' calculate the ratio of the stop-band frequency to the pass-band frequency of either Butterworth or Tchebysheff filters. These are useful when you have specific stop-band attenuation requirements, and are trying to determine what order of filter is required.

```
Fs←BOMEGAS params -- Computes Normalized Butterworth Stopband Frequency
'params' is a 3 element vector:
     params[1] = Number of poles
     params[2] = Max Passband Atten (dB)
     params[3] = Min Stopband Atten (dB)
'Fs' is the normalized Stopband edge
User is prompted for input if 'params' isn't 3 elements


Fs←COMEGAS params -- Computes Normalized Tchebysheff Stopband Frequency
'params' is a 3 element vector:
     params[1] = Number of poles
     params[2] = Ripple (dB)
     params[3] = Min Stopband Atten (dB)
'Fs' is the normalized Stopband edge
User is prompted for input if 'params' isn't 3 elements
```

7.11 MISCELLANEOUS FILTER FUNCTIONS

There are three remaining functions in *LLFILTER*: '*ACTIVEFLT*' for designing op-amp active filters, '*DIPLEXER*' for designing LC diplexers and '*TAPL*' for designing tapped inductor impedance transformation networks.

```
net←ACTIVEFLT params -- Models a 2-Pole Low-Pass Active Filter
Uses MARTHA 'OPAMP' element to implement the filter on p 289 of
"Operational Amplifiers" by Tobey,Graeme,Huelsman.
'params'= Ho(Negative),Fo(Hz),α(1.414 for Butterworth),C2(Farads)
Ho is the DC gain, and α is 2 times the damping coefficient.


net←'LPorHP' DIPLEXER params -- Synthesizes Lumped-Element Diplexers
'LPorHP' defines the response desired: 'LP' gives a network with the
High-Pass section terminated, 'HP' terminates the Low-Pass section.
'params' is a four element vector:
       params[1] = Ripple (dB), (0 dB is invalid)
       params[2] = Number of poles (per section)
       params[3] = Crossover frequency (Hz)
       params[4] = Source and Load Resistance (Ohms)
'net' is a MARTHA network with ZG and ZL specified.
Prototype design is generated by background fn 'prot1'
Fn prompts for inputs if response type or any of 'params' are missing


net←TAPL params -- Computes tapped inductors corrected for loaded Q
Computes the tapped-inductor required to perform a step-down impedance
transformation, including the effects of loading.  'net' is a MARTHA
network of L1 and L2.  'params' = RL(Ohms),RP(Ohms),LP(nH),Fo(MHz)
Function prompts for inputs if any are missing.
NOTE: RP÷RL > 2
```

```
Z→ o--o----              Z→ o--ΩΩΩΩ--o----
    |    |                   L1  |    |
    c    >    ----\           c    >
LP  c    >  RP ----/      L2  c    >  RL
    c    >                    c    >
    |    |                    |    |
    ∇    ∇                    ∇    ∇
```
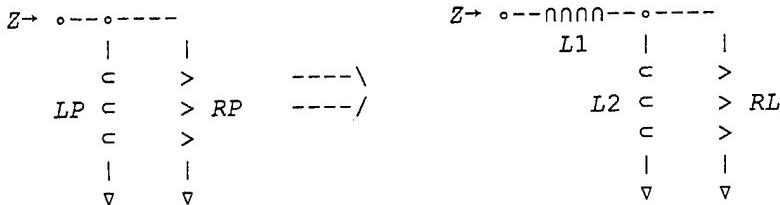
7.12 BACKGROUND FUNCTIONS

In addition to the functions listed above, there are a number of background functions (with lower case names) that are called by these functions, but are not typically executed directly by the user. If you need to create a specialized filter synthesis routine, you may want to investigate these. Many filter designs begin with the calculation of prototype 'g' values, or 'k' and 'q' values, and functions already exist to compute these. In addition, *MARTHA* includes a number of functions to do frequency and impedance scaling of networks to further simplify the task.

Below are the *EXPLAIN* listings for the three more useful background functions in *LLFILTER*:

```
kq←kqvalues Ripple(dB),Npoles -- Background Fn for Filter Synthesis
This function determines normalized 'k' and 'q' values for four
filter types: 1) Butterworth (Ripple=0), 2) Tchebysheff (Ripple>0),
3) Constant K (Ripple=⁻1), and 4) Gaussian (Ripple=⁻2), for any number
of poles.  The generator impedance (ZG) is 1 Ohm, and the bandwidth
(ripple BW for Tchebysheff, 3 dB for others) is 1 Radian.
'kq' is an (Npoles+1) element vector:
    kq←q1,k12,k23,- - -,k(Npoles-1)Npoles,qNpoles
Note: Constant K (q1←k12←k23 - - qN←1) produces minimum loss for a
given stop-band.

net←'LorC' proto Ripple(dB),Npoles -- Background Fn for Filter Synthesis
Synthesizes a low-pass prototype Butterworth or Tchebysheff filter of
any number of poles.  'LorC'='L' or 'C', Indicates end element type.
If 'Ripple'=0, response is Butterworth, with the 3 dB BW = 1 Radian.
If 'Ripple'≠0, response is Tchebysheff, with the ripple BW = 1 Radian.
'net' is a MARTHA network description with 'ZG'=1 Ohm, and 'ZL' set
to the required value depending on the filter specs.

elements←protog Ripple(dB),Npoles -- Background fn for BPXFMR
Synthesizes standard doubly-terminated ladder-type Butterworth or
Tchebysheff low-pass filters of any number of poles.  'elements' is a
vector of normalized element values for a 1 Radian bandwidth and 1 Ohm
generator impedance.  If Ripple=0, filter is Butterworth and BW is
3 dB.  For Ripple≠0, filter is Tchebysheff, and ripple BW is used.
Refer to Matthaei,Young,Jones 'Microwave Filters etc.', p. 95 for
circuit topology.  'elements' is equivalent to g1, g2, g3, etc.
```

7.13 ACKNOWLEDGMENTS

Most of the functions in *LLFILTER* were originally developed by Dave Hodsdon.

8. *LLPHASE* WORKSPACE

MARTHA COMPATIBLE PHASE DISTORTION ANALYSIS

8.1 INTRODUCTION

One concern when developing filters for RF systems is phase distortion. This may be described as group-delay variations or deviation from linear phase. While working on some filters for a receiver design, several *APL* functions were written to both analyze the phase distortion of the filters and synthesize all-pass compensating networks to correct for the most common type of distortion. To illustrate the use of these functions, an elliptic band-pass filter centered at 110 MHz will be used. The response of the filter is shown in Figure 8.1.

TRANSMISSION AND REFLECTION RESPONSE OF 'ELBPNET'

Figure 8.1. Frequency response of 3rd order elliptic filter

The functions in this workspace are designed to work with *MARTHA*. Loading *LLPHASE* automatically copies the *LLMARTHA* workspace, which is assumed to be in Library 8. If you are using a different library for the *LLAMA* workspaces, you with need to edit the argument of the *COPY△WS* command in the *STARt* function as described in Sections 1.8 and 2.2.

8.2 GROUP DELAY ANALYSIS FUNCTIONS

One problem with elliptic filters is that they exhibit the most phase and group delay distortion of any of the standard filter types. A typical task would be to calculate the group-delay of such a filter. There is a group-delay function, *AGD*, distributed with *MARTHA* in the *MARTHAD* workspace that calculates the approximate group-delay of a network. Figure 8.2 shows the group-delay of the filter calculated using *AGD*.

Figure 8.2. Group-Delay using AGD function

Unfortunately, the plot is dominated by the 'glitches' produced by the zeroes of the elliptic filter. This is 'accurate', in the sense that the group-delay isn't really well defined at these points, but it produces a result that is difficult to work with. There is another function, called *GD*, that was written to deal with group delay plots in a somewhat cleaner fashion. The result of running the *EXPLAIN* function on *GD* is listed below:

```
    EXPLAIN 'GD'
Out←GD Net -- Computes Approx Group Delay of a MARTHA Network
Computes (ΔTheta/ΔF)÷¯2×pi for a 0.1% ΔF if F is 1 point.  A ΔF of
0.1×(F[i]-F[i-1]) is used as a starting point if F is a vector.
ΔF is reduced until ΔTheta ≤ 0.175 Rad (10 Deg).  The transfer
parameter used is S21, and 'Out' is a MARTHA FOF.  Fn is loosely
based on an old Mainframe fn, but has been extensively cleaned up
and enhanced for speed.  NOTE: The algorithm 'searches' for smooth
phase, and will 'avoid' singularities (i.e. from zeroes of ellip.
filters).  Results are more usable than a rigorous answer with the
spikes such singularities produce.
```

Figure 8.3 shows the group-delay response of the same elliptic filter computed using *GD*. As long as you are aware that the singularities are still there, the results from *GD* are scaled in a manner that allows much easier study of the critical regions of the plot. Because it is so poor, the group-delay of elliptic filters isn't treated much in the literature, so it isn't clear how such singularities are typically handled.

52
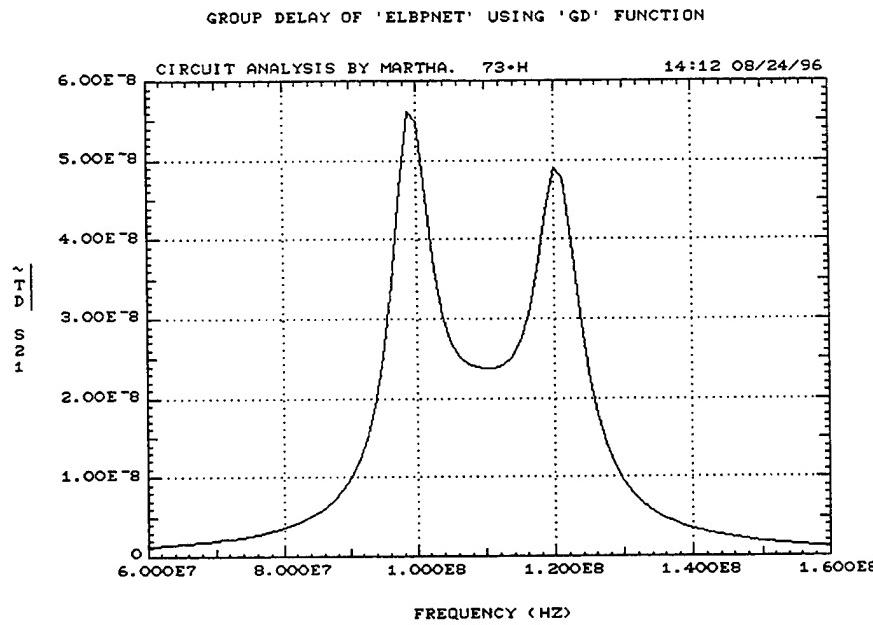
Figure 8.3. Group-Delay response using GD function

8.3 PHASE NON-LINEARITY FUNCTIONS

A common response used for this is the 'deviation from linear phase'. This is merely the phase of a circuit with a constant delay term subtracted out. The function *PHASNL* was written to calculate this, and the *EXPLAIN* listing for this is shown below:

```
EXPLAIN 'PHASNL'
Out←Fo(Hz) PHASNL Net -- Computes Deviation From Linear Phase
Computes the phase of MARTHA network, with the linear phase at
Fo subtracted out.  A ΔF of 0.1×(F[i]-F[i-1]) is used as a
starting point to determine the phase slope at Fo.  ΔF is reduced
until ΔTheta ≤ 10 Deg.  Uses the fn PHUNWRAP to obtain the
unwrapped phase of S21 of the network.  'Out' is a MARTHA FOF.
```

As mentioned above, *PHASNL* uses a function called *PHUNWRAP*. This is a useful function for a variety of applications. The *EXPLAIN* listing for *PHUNWRAP* is:

```
EXPLAIN 'PHUNWRAP'
Angl←PHUNWRAP Net -- Unwraps The Phase of S21 of a MARTHA network
The phase unwrapping is done by rotating the phase vector 1 point,
and then subtracting.  If any point of the result is > 0, a wrap
must have occurred, and -360 Degrees is added to all subsequent
points.  Result is a MARTHA FOF.
```

Figure 8.4 shows the unwrapped phase of the filter from *PHUNWRAP*, and Figure 8.5 shows the phase distortion analyzed using *PHASNL*. Note that the singularities introduce 180° phase jumps, which are not removed by the unwrapping process.

53

UNWRAPPED PHASE OF 'ELBPNET'
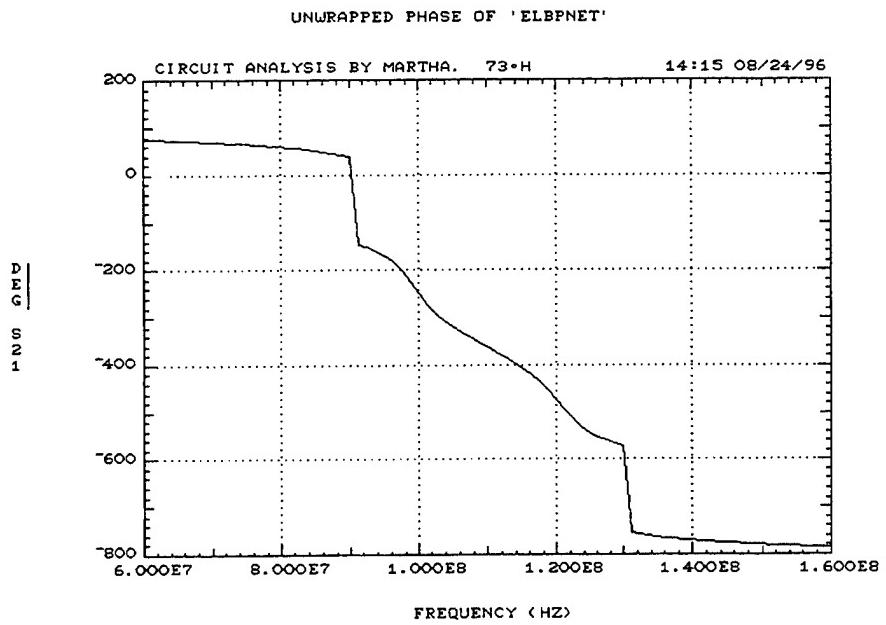
Figure 8.4. Unwrapped phase of elliptic filter

DEVIATION FROM LINEAR PHASE OF 'ELBPNET'
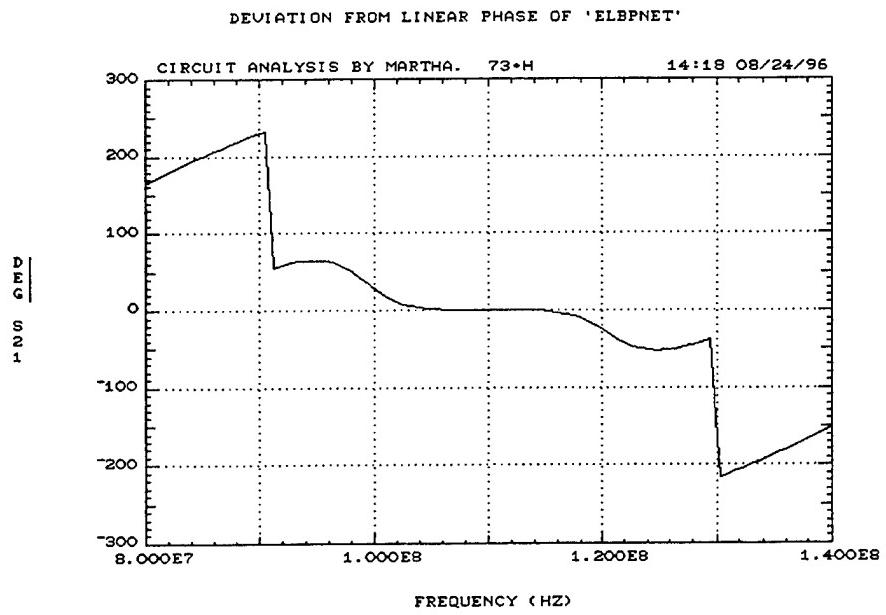
Figure 8.5. Deviation from linear phase of elliptic filter

8.4 PHASE COMPENSATION WITH AN ALL-PASS NETWORK

Over a ±10 MHz bandwidth, the phase deviation is ~±30°. This is more than enough to introduce significant distortion in some communications systems. In order to reduce this, a phase compensator can

54

be added. A second-order all-pass circuit can be used, which the function *ALLPASS* synthesizes. The *EXPLAIN* listing is shown below:

```
EXPLAIN 'ALLPASS'
net←Zo ALLPASS Fo(Hz),Q -- Creates A 2nd Order All-Pass Network
Creates a MARTHA network description of the following circuit:
                   La
        |------nnnnn------|
        |                 |
        |   Ca         Ca  |
    O--o---||---o---||---o--O        Note: Q must be > 1
                |
               === Cb
                |
    Zo →        ⊃              ← Zo
                ⊃ Lb
                ⊃
                |
    O------------o-----------O
```
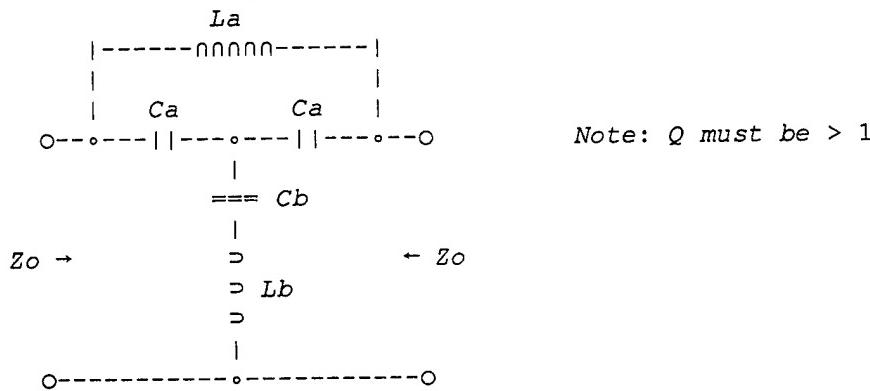
Circuit can be used to provide Group Delay equalization for filters.
REF: Williams, A.B., Taylor, F.J., 'Electronic Filter Design
Handbook' McGraw Hill, 1988, pp 7-4 to 7-6

The result of adding an all-pass network to the filter is shown in Figure 8.6. After a few cut-and-try experiments, the best performance was achieved with a Q of 5. The phase ripple is now ~±6° (note the change in vertical scale).
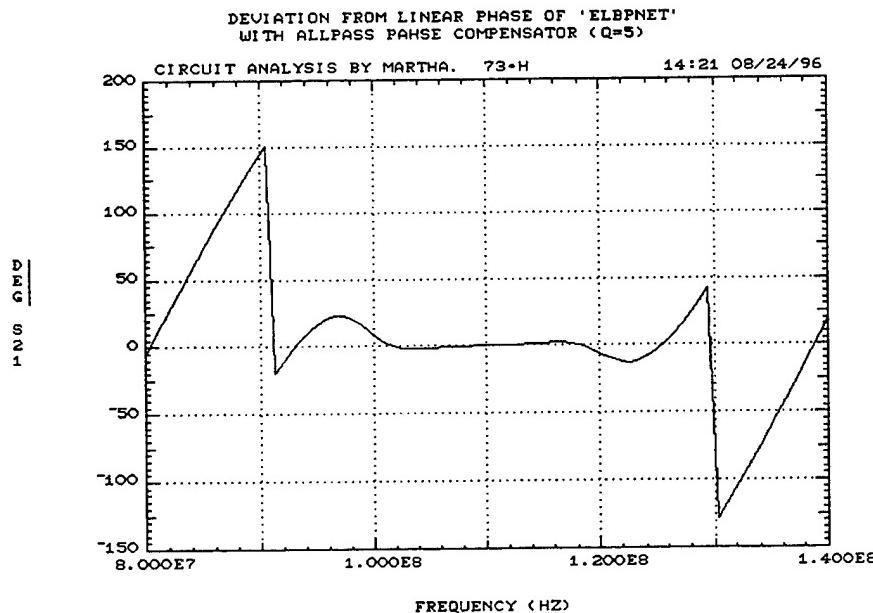
Figure 8.6. Elliptic filter with all-pass phase compensator

55

8.5 ACKNOWLEDGEMENTS

9. *LLMIXER* WORKSPACE

MIXER SPUR ANALYSIS

9.1 INTRODUCTION

One of the more demanding tasks in designing RF systems is coming up with a 'clean' frequency plan. This requires selecting RF, IF and LO frequencies so that unwanted mixer products do not get folded back into a signal band somewhere. The *LLMIXER* workspace contains functions to provide a graphical analysis of mixer products and spurs to aid in this process. *LLMIXER* has been documented to work with the *EXPLAIN* and *SUMMARY* functions, and follows the various programming conventions of other *LLAMA* workspaces.

There are two main functions in the *LLMIXER* workspace: *MIXSPUR*, and *SKIPSPUR*. *MIXSPUR* calculates and plots mixer spurs given a Local Oscillator frequency, the RF and IF bands, and the maximum order to check. It will also optionally highlight an RF/IF sub-band. *SKIPSPUR* is identical to *MIXSPUR*, except that it allows omitting specific spurs from the plot.

Both functions will prompt for inputs if called with an empty vector (' ') as a right argument. All frequencies are specified in MHz. *SKIPSPUR* can be used to sort out particularly busy spur plots, if some spur labels get overwritten by other spurs. If the frequency plan and bands are specified in 'round' numbers, some spurs may just 'nick' the corner of the plot. These spurs are labeled with a single letter at the appropriate corner. Once a plot is on the screen, typing <shift>P will exit and produce a hardcopy. Pressing any other key will exit without printing.

Here is a copy of the *EXPLAIN* result for *MIXSPUR*:

```
    EXPLAIN 'MIXSPUR'

[Band] MIXSPUR params -- Plots Mixer Spurs For Given LO,RF,IF and Max Order
'params' is a six element vector (all Frequencies in MHz):
  params[1] = Local Oscillator Frequency
  params[2] = Lower End of RF Band
  params[3] = Upper End of RF Band
  params[4] = Lower End of IF Band
  params[5] = Upper End of IF Band
  params[6] = Maximum Order (M+N) To Be Checked
Function prompts for input if any parameters are missing.
'Band' is an optional frequency range (Lower Limit, Upper Limit) which
when combined with the LO, defines a box on the plot.  The frequency
band is assigned depending on whether it is a subset of the IF or
RF band.  Function prompts for 'Band' only if it has prompted for
'params'.  'Shift-P' dumps screen to printer.
```

SKIPSPUR is identical except for the extra parameters to indicate which spurs to skip:

EXPLAIN 'SKIPSPUR'

[Band] SKIPSPUR params -- Similar To MIXSPUR, But Can Omit Specified Spurs
'params' is an eight or more element vector (all Frequencies in MHz):

        params[1]   = Local Oscillator Frequency
        params[2]   = Lower End of RF Band
        params[3]   = Upper End of RF Band
        params[4]   = Lower End of IF Band
        params[5]   = Upper End of IF Band
        params[6]   = Maximum Order (M+N) To Be Checked
        params[7]   = Order of LO for 1st spur to skip (0 to see all spurs)
        params[8]   = Order of RF for 1st spur to skip (0 to see all spurs)
     params[7+2J] = Order of LO for Jth spur to skip
     params[8+2J] = Order of RF for Jth spur to skip
Function prompts for input if any parameters are missing. The order
of spurs to skip must be entered in pairs. 'Band' is an optional
frequency range (Lower Limit, Upper Limit) which when combined with
the LO, defines a box on the plot. The frequency band is assigned
depending on whether it is a subset of the IF or RF band. SKIPSPUR
prompts for 'Band' only if it has prompted for 'params'. 'Shift-P'
dumps screen to printer.

Below is a sample plot showing the highlighted sub-band, and the legend indicating that one spur has been skipped:
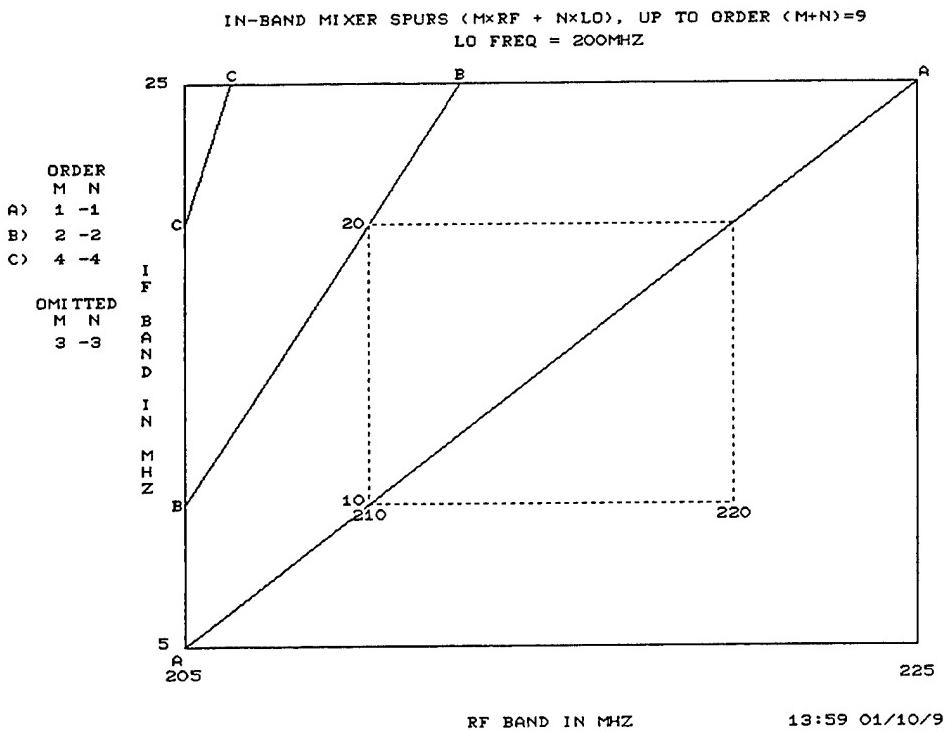
Figure 9.1. Example spur plot

The plotting functions used in this workspace are based on an old version of the '□G' graphics. These functions ARE NOT compatible with the newer □G plotting workspaces, so this should be kept separate. The default display device is set for VGA. If you have an EGA display, you must edit the 'init' function (lines 4 and 5). The printer is currently set to be an HP LaserJet. If you have an Epson LQ type printer, edit lines 45 and 46 of the 'plotspur' function. If you are running the program under a Windows operating system, you can always capture the results by copying the screen to the clipboard using Alt-PrintScreen.

9.2 'RUN TIME' VERSION OF *APL+PC* MIXER SPUR PLOTTING SOFTWARE

One of the unique features of APL2000's *APL+PC* is that you can create stand-alone 'run-time' programs. These are slightly modified workspaces that have been processed with a 'run time interpreter' to create a DOS executable program. This allows users without an *APL* interpreter to run programs written in APL. The process is best suited to well defined-single purpose workspaces like *LLMIXER*. If the *APL* workspace is modified to work with standard ASCII characters (converting the *APL* high minus '¯' to a regular ' - ', for example), the user need never know about *APL*, the keyboard, character set etc. The resulting program can be now run on any IBM PC in either DOS or Windows. Although programs run a bit slower when executed in this manner, they are perfectly functional.

Starting from the *LLMIXER* workspace, the program MIXSPUR.EXE was created using this system, and is included with the *LLAMA* files. In order to have one program rather than two, it is based on the *SKIPSPUR* function, with an option of displaying all spurs. The program is 'hardwired' for a VGA display, and an HP LaserJet compatible printer. A sample session is show below, using the same input parameters that generated the plot in Figure 9.1. If you look closely, you will see that the spur to be skipped (3 -3) was designated using the standard minus sign.

```
     Lincoln Laboratory Advanced MARTHA Applications (LLAMA): MIXSPUR Program
             Copyright 1996 by the Massachusetts Institute of Technology

MIXSPUR calculates and plots mixer spurs given a Local Oscillator frequency,
the RF and IF bands, and the maximum order to check.  It will also optionally
hi-lite an RF/IF sub-band.  The program can omit specific spurs from a plot if
things get too cluttered.  The program will produce a screen plot on a standard
VGA display, and can print a hardcopy on any HP LaserJet compatible printer.

The 'Sub-Band' is an optional frequency range (Lower Limit, Upper Limit)
which when combined with the LO, defines a box on the plot.  The frequency
band is assigned depending on whether it is a subset of the IF or RF band.

The spur-skipping feature can be used to sort out particularly busy spur
plots, if some spur labels get overwritten by other spurs.  If the frequency
plan and bands are specified in 'round' numbers, some spurs may just 'nick'
the corner of the plot.  These spurs are labeled with a single letter at the
appropriate corner.  Once a plot is displayed, typing <shift>P will produce
a hardcopy.  Pressing any other key will return from the plot without printing.
The program will then ask if you wish to exit or run another analysis.
```

★★★★★★★★★★★★★★★★★★★★★★★★★ *MIXSPUR* ★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★

LO, RF(low), RF(high), IF(low), IF(high), Maximum Order (Freqs In MHz):
200 205 225 5 25 9

Order of spurs to skip in M,N pairs (MxRF,NxLO), 0 0 shows all spurs:
3 -3

RF or IF Sub-Band to be hi-lited (in MHz), or <ENTER> for none:
10 20

At this point, the program draws the plot and pauses. When you leave the plot, the program asks:

Do You Want Another Plot? (Y/N): n

9.3 ACKNOWLEDGMENTS

This was loosely based on an old mainframe program by Dave Hodsdon, but with considerable enhancements. Bob Actis provided the motivation for creating the run-time version.

10. *LLTIME* WORKSPACE

MARTHA COMPATIBLE TIME DOMAIN ANALYSIS

10.1 INTRODUCTION

One of the *APL* workspaces that has been used for a number of systems is the *LLTIME* workspace. This contains functions to perform time domain analysis on linear circuits. *LLTIME* uses the *APL FFT* and *IFFT* functions developed by Prof. Paul Penfield Jr. at MIT. All of the functions have been heavily commented, and have been set up to work with the *SUMMARY* and *EXPLAIN* functions. They are also designed to be compatible with the *LLMARTHA* plotting routines, and the network analysis functions automatically produce X and Y labels for plotting using the *MARTHA* 'hd' header variable.

Because this workspace is primarily intended to be used with *MARTHA*, loading *LLTIME* automatically copies the *LLMARTHA* workspace, which is assumed to be in Library **8**. If you are using a different library for the *LLAMA* workspaces, you will need to edit the argument of the *COPY*Δ*WS* command in the *STARt* function as described in Sections 1.8 and 2.2.

10.2 TIME DOMAIN ANALYSIS FUNCTIONS

The two most useful functions are '*AMPLVST*' and '*IFAMPVST*', which calculate the time response of a *MARTHA* network to the waveform shown in Figure 10.1. *AMPLVST* operates at baseband using the circuit in Figure 10.2, and *IFAMPVST* converts the response around some IF down to baseband using the circuit in Figure 10.3. There is also an '*IMPULSE*' function, which computes the impulse response of a network.
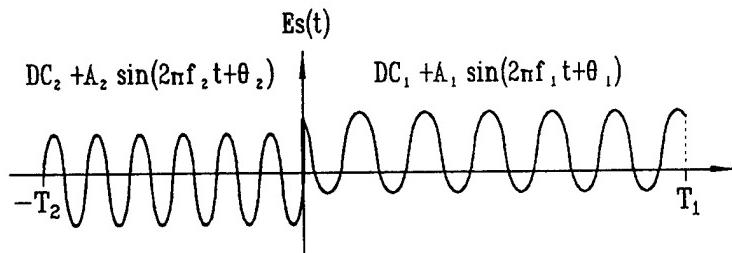
Figure 10.1. Input waveform for time-response functions
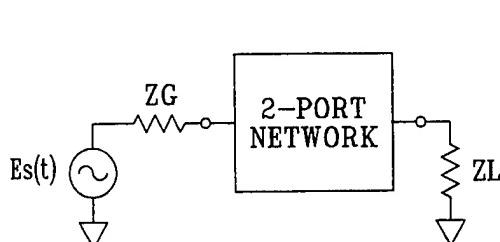
Figure 10.2. System analyzed by AMPLVST   *Figure 10.3. System analyzed by IFAMPLVST*
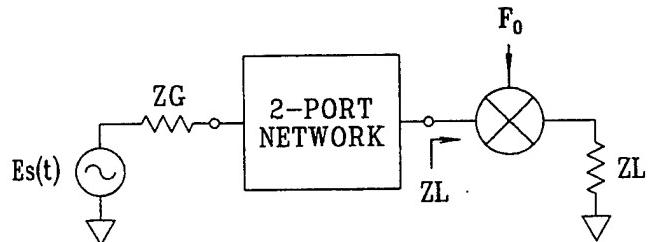
```
Mat←Net AMPLVST params -- Computes Complex Time Response of a MARTHA Network
The input waveform consists of a sequence of two sine waves, each of
arbitrary amplitude, frequency, phase, DC offset, and duration.  Mat has
4 columns;  Real, Imaginary, Input, and Time.


Mat←Net IFAMPVST params -- Computes Time Resp. of a MARTHA Network at an IF
Similar to AMPLVST, only the response is calculated about some IF, and then
converted to baseband.  Mat has 4 columns; Real, Imaginary, Input, and Time.


Mat←Net IMPULSE Npts,T -- Computes Complex Impulse Resp of a MARTHA Network
Mat has 3 columns; Real, Imaginary, and Time.
```

The parameter list for the right argument of *AMPLVST* and *IFAMPVST* is pretty involved, and both functions will prompt the user if called with an empty vector. There are several examples demonstrating the use of these functions and the input parameters later in the section. The left argument '*Net*' is a character vector description of a *MARTHA* network, and the result '*Mat*' is a multi-column matrix with time as the last column. This allows the results to be plotted easily without re-formatting. The last result calculated by any of the 3 main time response functions is stored in the global variable '*TAME*' for later use.

10.3 OUTPUT MODIFIER FUNCTIONS

There are five additional functions that work with the network response functions and the '*SPECTRUM*' function (described below). These allow you to extract various types of information from the output of the analysis functions, i.e. the envelope of a response, the real or imaginary part, etc.

```
dBV←DBV A -- Converts Output From Time Response Functions To dBV
Output is 2 columns; dBV, and Time.


Env←ENV A -- Calculates the Envelope of Complex Time Response Signals
Output is 2 columns; Envelope, and Time.  NOTE: Doesn't work on SPECTRUM


Real←REAL A -- Picks Off Real Part of Complex Time Response Signals
Output is 2 columns; Real Part, and Time.


Imag←IMAGINARY A -- Extracts Imaginary Part of Output Time Response Fns
Output is 2 columns; Imaginary Part, and Time.


MagV←MAGV Signal -- Converts Complex Time Response To Magnitude w/Sign
Calculates the magnitude of the signal, but restores the sign so that
a bipolar waveform results.  Output is 2 columns; 'Magnitude', and Time.
```

10.4 TIME RESPONSE EXAMPLES

The first example shows the response of a low-pass filter to a rectangular pulse (the input frequencies are zero, and the pulse is created by setting the phase of one signal to 90 degrees). The network is:

LPNET
(C 4.5E⁻9) WC WS L 11.25E⁻6

Figure 10.4 was created using an empty argument to get prompts from *AMPLVST*:

```
PLOT REAL LPNET AMPLVST ''
1st Signal Params T1(secs),A1(volts),θ1(degs),F1(Hz),<DC1(Volts)>:
0.5E⁻6 2 90 0
2nd Signal Params T2(secs),A2(volts),θ2(degs),F2(Hz),<DC2(Volts)>:
1.5E⁻6 0 0 0
Enter Number of Points (Pwr of 2): 256
```
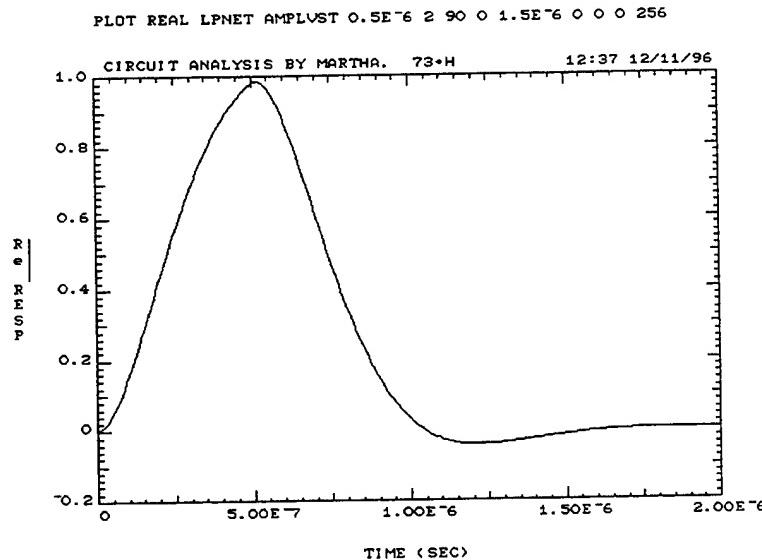

Figure 10.4. Time response of a square pulse through a low-pass filter

The next example analyzes the response of a 25 MHz bandpass filter:

BPNET
((C 2.25E⁻9)P(L 1.801E⁻8)S R 0.01)WC WS (L 5.627E⁻6)S(C 7.2E⁻12)S R 0.01

Figure 10.5 was created by:

```
PLOT REAL BPNET AMPLVST ''
1st Signal Params T1(secs),A1(volts),θ1(degs),F1(Hz),<DC1(Volts)>:
0.5E⁻6 2 0 25E6
2nd Signal Params T2(secs),A2(volts),θ2(degs),F2(Hz),<DC2(Volts)>:
1.5E⁻6 0 0 25E6
Enter Number of Points (Pwr of 2): 256
```

63

PLOT REAL BPNET AMPLUST 0.5E‾6 2 0 25E6 1.5E‾6 0 0 25E6 256
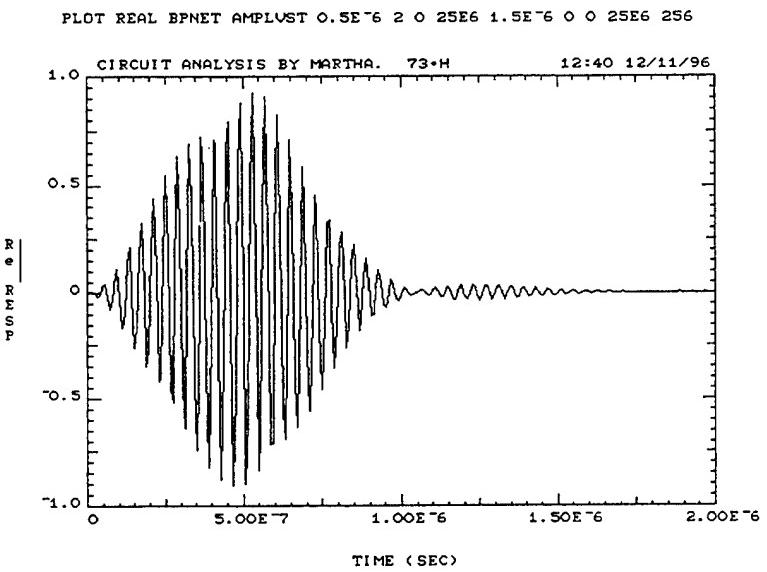
Figure 10.5. Time response of a tone burst through a band-pass filter

The next example was created using the data from the previous analysis, which is stored in the global background variable *TAME*, by running:

PLOT ENV TAME

PLOT ENV BPNET AMPLUST 0.5E‾6 2 0 25E6 1.5E‾6 0 0 25E6 256
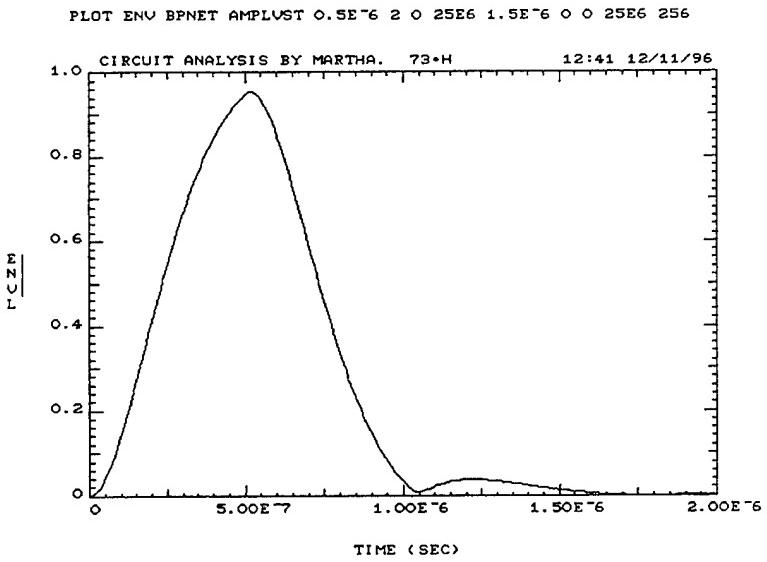
Figure 10.6. Envelope of a tone burst through a band-pass filter

10.5 FREQUENCY SPECTRUM FUNCTION

The function '*SPECTRUM*' computes the spectrum of a time waveform using the FFT function. The last result of running *SPECTRUM* is stored in the variable '*FAME*' for later use.

```
Mat←SPECTRUM V(t) -- Computes Spectrum of a Time Waveform Using FFT
V(t) is a 3 column matrix of Real, Imaginary, Time.  The Imaginary column
is optional for pure real signals.  Mat has 3 columns; Real, Imaginary,
and Frequency.
```

10.6 FOURIER SERIES FUNCTIONS

There are two functions, '*FOCO*' and '*FOSER*', for calculating and reconstructing Fourier Series representations of periodic waveforms. *FOCO* computes the Fourier coefficients of a periodic waveform, and *FOSER* reconstructs a time waveform from a set of Fourier coefficients.

```
Cmat←Nc FOCO Data -- Computes Fourier Series Coeffs of Periodic Waveforms
Produces a table of 'Nc' Fourier Series coefficients in Magnitude/Angle
format from a vector of one period of a time waveform.

Mat←Npts FOSER Coeffs -- Reconstructs Waveform From Fourier Series Coeffs
Checks the results of the FOCO function.  Can be used with PLOT as follows:

  PLOT FOCOdata AND (ρFOCODATA) FOSER Ncoeff FOCO FOCOdata
```

Figure 10.7 shows the Fourier Series reconstruction of a square wave from the 1st eleven harmonics using the *FOCO* and *FOSER* functions. The command string and data array are described in the plot title.
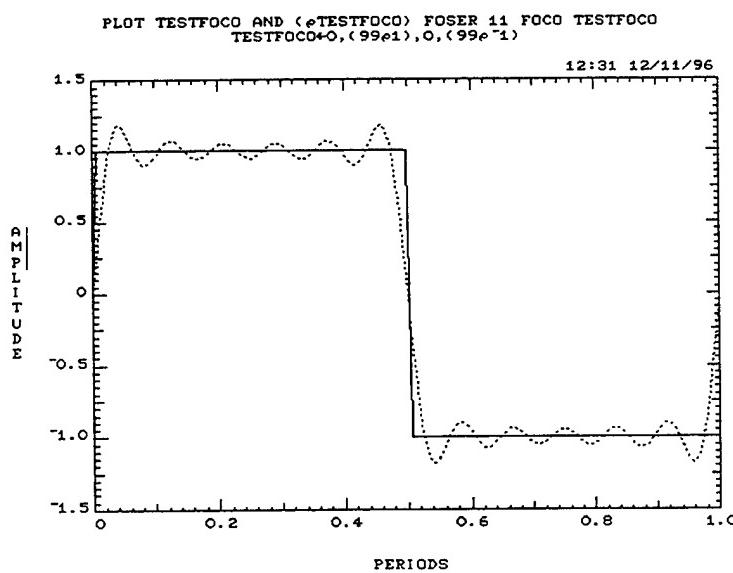
Figure 10.7. Fourier series decomposition of a square wave up to 11th harmonic

10.7 FAST FOURIER TRANSFORM (FFT) FUNCTIONS

Although typically not called by the user (thanks to the response functions mentioned above) the *EXPLAIN* listings for the two FFT functions at the heart of this workspace are given below:

```
Z←FFT X -- APL-Style Fast Fourier Transform
'X' can be real (an N element vector) or complex (2 row × N column
matrix).  Result is always a 2 row complex matrix.  N must be an
integral power of 2.  A global weight matrix 'wt' is created each
time N changes, but 'wt' is re-used for subsequent transforms of the
same length.  Function is optimized for speed.
Ref: Paul Penfield Jr., "EFFICIENT APL FAST FOURIER TRANSFORM",
LL Tech. Memo. No. 63L-0019, 5/16/80


Z←IFFT X -- APL-Style Inverse Fast Fourier Transform
'X' can be real (an N element vector) or complex (2 row × N column
matrix).  Result is always a 2 row complex matrix.  N must be an
integral power of 2.  A global weight matrix 'wt' is created each
time N changes, but 'wt' is re-used for subsequent transforms of the
same length.  Function is optimized for speed.
Ref: Paul Penfield Jr., "EFFICIENT APL FAST FOURIER TRANSFORM",
LL Tech. Memo. No. 63L-0019, 5/16/80
```

10.8 ACKNOWLEDGMENTS

In addition to the FFT functions by Prof. Penfield, the majority of the network time analysis functions were originally developed by Dave Hodsdon. Mark Stevens suggested adding DC offsets to *AMPLVST*, and a short-cut to reduce some of the network computations.

11. *MARTHAD* WORKSPACE

MARTHA 'DEVELOPMENT' WORKSPACE

11.1 INTRODUCTION

The '*MARTHAD*' (the 'D' stands for Development) workspace contains a number of functions that are not as well tested or documented as the regular *MARTHA* functions. Some have been in regular use for some time, and some of them may be a bit buggy. If you find something in here you desperately need and either have trouble understanding how to use it, or encounter problems when running it, contact your software vendor. Because these functions were written not long after the original release of the *MARTHA* Addendum, they weren't designed to work with the *EXPLAIN* documentation system. Instead, each function has a '*HOW*' variable that describes its usage.

11.2 NODAL WIRING FUNCTIONS

The functions '*WA*' and '*WF*', along with the background function '*RDE*', allow 'floating' networks to be connected into *MARTHA* circuits. A floating network is a network that can't be described as a connection of 2-ports, and must be described in a nodal-connection form. These functions are at the core of the combline filter synthesis workspace '*LLCOMBFL*' described in Section 6. The variable '*HOWNODAL*' contains a complete description.

```
            VARIABLE 'HOWNODAL' IN MARTHAD WORKSPACE

NODAL WIRING FUNCTIONS:  The functions 'WA' and 'WF' along with the
background function 'RDE', allow 'floating' networks to be connected
into MARTHA circuits.  A floating network is a network that can't be
described as a connection of 2-ports, and must be described in a
nodal-connection form.

For example, suppose you wish to create the following circuit:

                      24
        |------------||------------|
        |                          |
   [1]  |  20   [5]   1       50    |  [2]
    O--o--nnnn--O--vvvv--O--nnnn--o--O - - - -
             |           [¯3]
            === 4                      to other MARTHA 2-ports
             |
    O-------------------------------O - - - -
            [0]


 Node Numbering Conventions: Node [1] must be the input
                             Node [2] must be the output
                             Node [0] must be ground
                             Nodes can have negative node numbers


 'WF' is used to 'float' a network.  The syntax is:

     X  ← nn WF net
```
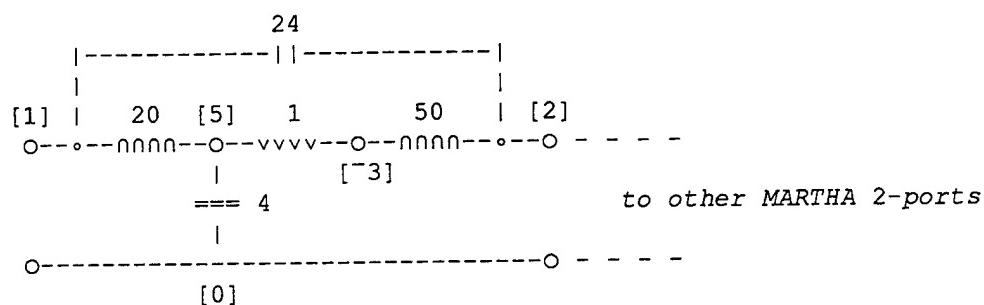
where 'net' is a standard MARTHA network, and 'nn' is a node number you assign according to the following conventions:

One-Port: nn[2]   O
```
                            _ _
                          _| |_
                         |     |
                         |     |
                         |_____|
                            |
```
                           nn[1]   O

3 Terminal:
```
                              _____
                             |       |
              nn[2]   O-|       |-O nn[3]
                             |_____|
                                 |
```
                              nn[1]   O
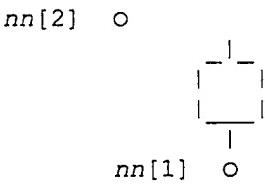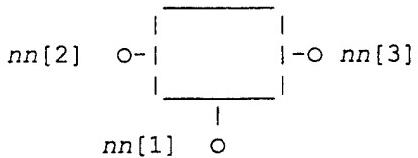
Two-Port:
```
                              _____
              nn[2]   O-|       |-O nn[3]
                             |       |
              nn[1]   O-|       |-O nn[4]
                             |_____|
```
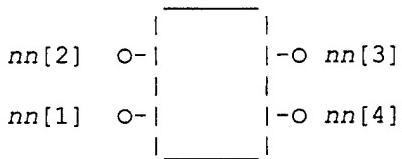
'WA' is used to connect 'floated' networks, for example:

N1 ← 1 5 WF L 20   ties the 20 H inductor between nodes [1] and [5]
N2 ← 5 ‾3 WF R 1   ties the 1 Ohm resistor between nodes [5] and [‾3]
N3 ← N1 WA N2 connects the two elements together

The entire nodal circuit description of the example above is:

NET ← N3 WA (5 0 WF C 4) WA (‾3 2 WF L 50) WA 1 2 WF C 24

This can now be incorporated in a regular MARTHA network:

NEWNET ← NET WC (WP R 50) WC WS C 10

Once NET is incorporated into a standard MARTHA 2-port connection the internal nodes are no longer available for use with the nodal wiring functions. The maximum number of 'elements' in one nodal description is limited by:

$$155 \leq (No. \ of \ 1\text{-}ports) + 4 \times (No. \ of \ 2\text{-}ports)$$

11.3 STRIPLINE MODEL

'STRIPLINE' is a MARTHA circuit element model for stripline transmission lines. The STRIPLINE element is used the same way as 'TEM' in a MARTHA network, and if length is omitted from the argument vector, the function will return the characteristic impedance of the line. The details are outlined in the 'HOWSTRIPLINE' variable.

VARIABLE 'HOWSTRIPLINE' IN MARTHAD WORKSPACE

'STRIPLINE' is a MARTHA circuit element model for stripline transmission lines. The function is monadic, and requires 5 arguments in the following order: W-strip width, B-ground plane spacing, T-strip thickness, L-length, and E-relative dielectric constant. All physical dimensions are in meters. Example:

T1 ← STRIPLINE W B T L FORDIEL E

The 'STRIPLINE' element is used the same way as 'TEM' in a MARTHA network. If you wish to know the impedance of a section of stripline,

> *STRIPLINE W B T FORDIEL E*

will return the characteristic impedance of the line.

11.4 DATA REDUCTION TWO-PORTS

The functions 'SDRT', 'YDRT', and 'ZDRT' are used to compute Data-Reduction Two-Ports from measurement data. This is a method of calculating an equivalent de-embedding circuit from measured data. The variable 'HOWDRT' contains details on how this is done.

Interpretation of arguments of SDRT, YDRT, AND ZDRT:

Whenever a measurement of 1-Port Impedance Admittance, or Reflection Coefficient is made, the point at which the measurement is made, and the location of the device are different. Call the 'effective embedding network' between the two 'EEN', and denote the device 'DEV' and the measurement (i.e., the device as referred to the measurement plane) 'MEAS'. Thus MEAS=EEN WT DEV.

The problem is to infer the device characteristics from the measurements, i.e. to find 'DEV' form 'MEAS'. Normally, 'EEN' is not known a priori. Note that unless the device is isolated from the measurement plane, the relation between device impedance and measured impedance is a complex, frequency-dependent bilinear transformation. Therefore its inverse exists, and we may define a 'Data-Reduction Two-Port' 'DRT' with the property that DEV=DRT WT MEAS. In fact, many such two-ports exist, and we need only find one.

Step 1: To find one such DRT, find at least three device calibrations with known impedances 'ZC'. These might include short circuit, open circuit, standard capacitances, or diodes with known biases. Substitute these, one at a time, for the device, and measure the corresponding impedances at the measurement plane 'ZMC'. Arrange the calibration impedances and the corresponding measured impedances in the form of a FOF with 4×N columns, where 'N' is the number of calibration impedances used (at least 3). The columns should be arranged in the form of 'n' blocks of four adjacent columns. Each block contains the calibration impedance (real and imaginary parts) in the first two columns and the measured impedance in the last two. That is, the form

is: RE ZC, IM ZC, RE ZMC, IM ZMC. Since there must be at least three such blocks, the entire FOF must have at least 12 columns. It may have 12, 16, 20, 24, etc. columns.

Step 2: The function 'ZDRT' in the workspace '100 MARTHAE' is used to create the 'DRT' form this FOF: DRT←ZDRT A. The result will be a numerically defined two-port element. It will be reciprocal (regardless of whether 'EEN' is or not) and have an arbitrary sign for Z12.

Step 3: This DRT is now used to process subsequent measurements. Thus if 'MEAS' is now a numerically defined 1-Port element, for example created by the function 'ZFOF', then the device is DRT WT MEAS; to display its impedance and admittance, for example,
    PRINT Z, Y OF DRT WT MEAS.

If admittance measurements are made instead of impedance, use the function 'YDRT' instead of 'ZDRT'. If the measurements are reflection coefficient instead of impedance, use the function 'SDRT' instead of 'ZDRT'. The result in all cases is a numerically defined 2-Port element, a Data-Reduction Two-Port.

If more than three sets of calibration measurements are used then the effects of experimental error are reduced somewhat.

The relation between 'DRT' and 'EEN' is not a simple one. There are many possible Data-Reduction Two-Ports, and one of them is ‾1 ZSCALE WN EEN which may be used in case 'EEN' is known. However, that network is not generally equal to 'DRT'. In the first place, it is non-reciprocal if 'EEN' is, and even if it is reciprocal, it may differ from 'DRT' in the sign of Z12, which for 'DRT' is chosen at random. Despite these differences, the Data-Reduction technique is accurate. Although it is tempting to conclude that some properties of 'EEN' may be inferred from 'DRT', or even that EEN=‾1 ZSCALE WN DRT, this is dangerous. The only legitimate use of 'DRT' is for inferring 'DEV' from 'MEAS'.

11.5 APPROXIMATE GROUP-DELAY

The function 'AGD' is a response function that computes an approximation to the group-delay of a MARTHA network. There are related functions, 'ADF' and 'ADW', which compute the approximate derivative with respect to F, or ω (2πF) respectively, and similar functions for FOF's. Details on these functions, are contained in the variable 'HOWAGD'.

VARIABLE 'HOWAGD' IN MARTHAD WORKSPACE

The function 'AGD' is a MARTHA response function which computes the approximate group delay of a network. The user must select a sufficient number of closely spaced frequency points to ensure a meaningful answer. NOTE: This function does not include phase un-wrapping, so there will be discontinuities if the phase of the network passes through multiples of 360 degrees.

The functions 'ADF' and 'ADW' are approximate derivatives with respect to
frequency, and ω (2 pi×F). These have the same limitations and
requirements as 'AGD'.

The functions 'FAGD', 'FADF' and 'FADW' perform similar operations on
FOF's.

11.6 RATIONAL FUNCTION OF S

The functions 'RAT', 'RATNE' and 'NEWELEMENT' are used to analyze and work with rational functions of 'S' (jω). RAT fits a rational function to the complex impedance contained in a two-column FOF. RATNE, and NEWELEMENT are used to analyze rational function responses. The variable 'HOWRAT' has more information.

VARIABLE 'HOWRAT' IN MARTHAD WORKSPACE

The function 'RAT' fits a rational function of 'S' to the data in a two
column FOF. The function is dyadic, with the following syntax:

   Z ← O . RAT FOF

'O' is normally a two element vector, with 'O[1]' being the order of the
numerator, and 'O[2]' being the order of the denominator. If 'O' is a
scalar or a single element vector, the value of 'O' is used for both
numerator and denominator.

'FOF' must be a two column FOF (representing the real and imaginary parts
of an impedance), and must have at least ⌈(+/O)÷2 frequencies.

The result 'Z' is a vector of length 5 + (total order of num. and den.).
The first two elements are just 'O', the order vector, the 3rd element is
'1' (the constant multiplier), and the remaining elements are the
coefficients of the numerator polynomial, followed by the coefficients of
the denominator polynomial.

The functions 'RATNE' and 'NEWELEMENT' are used to analyze the response of
a rational function using MARTHA.

11.7 FOURIER TRANSFORM

The function 'FT' implements a Fourier transform for use with MARTHA. This is not a standard FFT, and is slow, but has some flexibility not available with an FFT. 'HOWFT' contains more information.

VARIABLE 'HOWFT' IN MARTHAD WORKSPACE

The function 'FT' is a Fourier Transform function that computes the time
response of a FOF with fairly arbitrarily specified time and frequency
points. Unlike a regular FFT, the frequency points do not have to be
equally spaced, or even in order. The time points the response is
computed for must be placed in the variable 'T', and can also be unevenly
spaced.

71

11.8 S-PLANE ANALYSIS OPTION

This uses the background option function '*OPT*' to allow the analysis of *MARTHA* network responses with inputs having both real and imaginary parts, i.e. signals off the usual $j\omega$ axis. The real part of the frequency is assigned to the vector '*SIGMA*', and the combined frequency array can be examined using the function '*CF*'. The variable '*HOWSPLANE*' has the details.

```
           VARIABLE 'HOWSPLANE' IN MARTHAD WORKSPACE

   Using the S-Plane analysis option, the response of any MARTHA network can
   be analyzed off of the normal jω axis.  The real part of the stimulus is
   defined by the variable 'SIGMA', which must contain the same number of
   points as the imaginary part, which is the normal 'F' vector.  In
   addition, the function 'OPT' from this workspace must be copied into the
   active workspace.  The complex stimulus array can be listed using the
   function 'CF'.  The convention for 'SIGMA' is that it includes the factor
   2×pi.

       EXAMPLE:

           )LOAD MARTHA
           )COPY MARTHAD OPT CF

           SIGMA←(O2)×1 2            ⍝ Define Real Part of complex freqs

           PRINT CF DB S21 OF C 1E⁻3   ⍝ Ask for analysis

       CIRCUIT ANALYSIS BY MARTHA.   73∘H   10/7/91 13:57

              F          RE CF         IM CF         DB S21
           ----------- ----------- ----------- -----------
           1.000E0000  6.283E0000  6.283E0000  ⁻1.347E0000
           2.000E0000  1.257E0001  1.257E0001  ⁻2.614E0000

   If you are not using the S-Plane analysis temporarily, execution speed will
   improve if you remove the 'OPT' function from the workspace, and replace
   it with a variable 'OPT' set to zero (OPT←0).
```

11.9 ACKNOWLEDGMENTS

Although there may have been contributors whose names have been lost to posterity, Prof. Paul Penfield Jr. wrote most, if not all, of the *MARTHAD* functions.

12. *LLRADIAL* WORKSPACE

MARTHA RADIAL MICROSTRIP ELEMENT

12.1 RADIAL MICROSTRIP ELEMENT

The function '*RADIAL*', and the background functions '*J0*', '*J1*', '*Y0*', '*Y1*', and '*NEWELEMENT*' implement a *MARTHA* compatible 2-port model of a radial microstrip element. This would ordinarily have been included in the *MARTHAD* workspace, but it relies on a *NEWELEMENT* function which is specific to *RADIAL*, and would interfere with the *NEWELEMENT* function used to analyze and work with rational functions of 'S' (jω). The variable '*HOWRADIAL*' in the workspace contains the details.

```
            VARIABLE 'HOWRADIAL' IN RADIAL WORKSPACE

There are 6 functions used to implement the radial microstrip two-port:
'RADIAL', which is the main function, 'J0', 'J1', 'Y0', 'Y1' and
'NEWELEMENT', which are background functions used by 'RADIAL'.

'RADIAL' is a monadic function that behaves like a MARTHA element, and
expects an argument of length 5 containing (in order): Er of substrate,
the angle (in degrees) of the line, the substrate thickness (in meters),
the radius at the input end, and the radius at the output end (both in
meters). The result is a MARTHA 2-port element which can be wired in with
any other MARTHA elements to form a network for analysis.  If you want to
implement a radial stub, use the MARTHA wiring function 'WTO' to create
a 1-port.

'J0', 'J1', 'Y0' and 'Y1' are Bessel functions of a real argument,
calculated using power-series approximations.  They behave like monadic
APL scalar functions, and return a result in the same shape as the
argument.  For 'J0' and 'J1', the arguments may be positive or negative.
'Y0' and 'Y1' require positive arguments.  These 4 functions can be used
by themselves, are self-contained, and make no use of global variables.
They will work in either index origin.

'NEWELEMENT' is a special function that provide the 'hooks' into MARTHA
to allow 'RADIAL' to act like a MARTHA element.  WARNING!: 'NEWELEMENT'
is a function name that is not unique to RADIAL.  If you encounter
problems, make sure you have the correct 'NEWELEMENT' for use with the
'RADIAL' function.  For example, the analysis of rational functions
requires a different 'NEWELEMENT' function.

All of these functions must be present in the active workspace for 'RADIAL'
to work, and □IO=1 is required.

References:

'RADIAL': Ramo, Whinnery, and Van Duzer, page 453 ff

Bessel Functions: Abramowitz and Stegun, pp. 369-370
```

12.2 ACKNOWLEDGMENTS

13. *LLRFCOIL* WORKSPACE

INDUCTOR WINDING AND PARASITIC ANALYSIS

13.1 INTRODUCTION

This workspace contains a variety of functions for dealing with some of the more practical aspects of LC filter design. There are several functions for calculating the value of small inductors for various geometries, as well as functions for modeling various parasitic elements.

All of the functions have been documented to work with the '*EXPLAIN*' and '*SUMMARY*' functions. The functions are grouped below roughly according to use, and the details are listed using the *EXPLAIN* function.

Although many of the functions in *LLRFCOIL* are designed to be used by themselves, some of the functions require *MARTHA*. Loading *LLRFCOIL* automatically copies the *LLMARTHA* workspace, which is assumed to be in Library 8. If you are using a different library for the *LLAMA* workspaces, you will need to edit the argument of the *COPY*Δ*WS* command in the *STARt* function as described in Sections 1.8 and 2.2.

13.2 FUNCTIONS FOR SMALL INDUCTORS

There are three functions for calculating the value of different geometries of small inductors. These are useful for calculating parasitics as well as designing small inductors.

```
Ind(H) ← BARL L,W,T(Inches) -- Computes the Inductance of a Rect. Bar
Prompts user for inputs if ρL,W,T ≠ 3

L ← LOOP Radius,W(Inches) -- Inductance of Single Turn uStrip Loop
Use mean radius of loop.  Accurate for loop diameters < 1/10 wavelength
REF: 'Microwave Field-Effect Transistors--Theory, Design, And
      Applications', Pengally, Research Studies Press Ltd., P. 426

L(nH) ← Length(Inches) WIREL Dia(Inches) -- Inductance of a Short Wire
```

13.3 SINGLE-LAYER-SOLENOID (SLS) INDUCTOR FUNCTIONS

This is one of the most common inductor constructions for RF filters. The function '*SLSOL*' calculates the standard handbook formula for such an inductor. Of more practical use is the function '*COIL*', which uses AWG wire size and takes into account the thickness of the 'Nyleze' insulation commonly used on modern magnet wire. A very convenient and repeatable method for winding such coils is to use the shank of a numbered drill bit as a mandrel (form). The function '*DRILL*' converts a drill number into the appropriate diameter to be used by the *COIL* function. For example, you could execute

```
COIL (DRILL 27),24,0,12
```

to get the inductance of a coil wound on a #27 drill bit. The function '$TABLE\Delta COIL$' uses the $COIL$ function to create a table of inductance values over a range of wire sizes and number of turns, all wound on a given sized form (drill).

```
L(nH) ← Nturns SLSOL ID,WireD -- Computes Ind. of Single-Layer Solenoid Coil
ID = Inside or form diameter (inches)
WireD = Bare wire diameter (inches)
Nturns can be a vector

L(H) ← COIL Dia,W(AWG),Spc,Nturns -- Computes Inductance of Nyleze SLS
Computes inductance of single-layer-solenoid coils made with Nyleze wire.
Dia is diameter of form in inches, W is wire size, Spc is wire spacing
in inches.  Nturns can be a vector of numbers for multiple coils.
Function will prompt user if right argument is less than 4 in length.
Valid for wire sizes between AWG 10 and 50.  Requires the background
functions 'coilb' and 'coilk'.

Dia(Inches) ← DRILL DrillNo. -- Returns Diameter From Number Drill Size

TABLE∆COIL -- Creates a Table of Single-Layer-Solenoid Coils Using 'COIL' Fn
Function Prompts User for All Inputs.  Creates a table for a given
inner (form) diameter for various wire sizes and number of turns.
```

13.4 PARASITIC ANALYSIS FUNCTIONS

The remaining two functions can be used to model and analyze the effect of parasitics on your filter components. The first function '$COILQ$' is actually used to calculate the model elements of an inductor measured in series with a 50 ohm transmission line on a network analyzer. The second function plots the effect of parasitic inductance on the apparent value of a capacitor.

```
COILQ -- Computes Q, C, L etc. of Parallel Resonator In Series W/50 Ohm Line
Prompts user for Center Freq.(MHz), BW(MHz) and lowest S21(dB)
Assumed MARTHA model is:  WS (C C1) P (L L1) S R R1

RFCAP -- Plots Effective Capacitance of a Capacitor Near Resonance
Prompts user for Nominal Capacitance(pF) and Self-Resonance Freq(MHz)
```

13.5 ACKNOWLEDGMENTS

Most of these functions were originally written by Dave Hodsdon.

14. *LLDRANGE* WORKSPACE

DYNAMIC RANGE ANALYSIS AND PLOTTING

14.1 INTRODUCTION

The *LLDRANGE* workspace contains functions to tabulate and plot the dynamic range parameters of an RF system. There are functions to enter and edit the various parameters for each stage of the system, and functions to plot or print a detailed graph of the combined gain, noise performance, gain compression, etc. in a format that makes aids further analysis and design.

There are 4 main functions: *MAKEDATA, EDITDATA, DPLOT,* and *DPRINT. MAKEDATA* prompts the user to create an array of stage parameters and a vector of stage names. *EDITDATA* allows editing the stage parameters of an existing system. *DPLOT* calculates the cascaded parameters and displays a plot of the results along with tables. *DPRINT* uses the same inputs as *DPLOT*, but prints directly to a printer without requiring a high resolution screen driver.

The results of running *EXPLAIN* on the four main functions are given below:

```
    EXPLAIN 'MAKEDATA'
MAKEDATA 'Name' -- Creates Input Data Matrix of Stage Parameters
Automatically forms input data variables to do dynamic range plots.  If
'Name' is an empty vector, the user is prompted for a new system name, and
then for stage parameters.  If 'Name' doesn't exist, it will be used for a
new system.  If 'Name' already exists, the user will be prompted for data
to add stages after the last existing stage.  The plot resolution can
currently support up to 12 stages.

    EXPLAIN 'EDITDATA'
EDITDATA 'Name' -- Allows Editing Stage Parameters of an Existing System
'Name' must be the name of a dynamic range analysis system, entered as a
character vector.  If 'Name' is an empty vector, fn prompts for an existing
system name.  The system parameters are displayed, and the user is prompted
first for the stage and then the parameter to be changed.

    EXPLAIN 'DPLOT'
<User(s)> DPLOT 'Name' -- Plots Dynamic Range Analysis Data to Screen
This function performs a number of dynamic range calculations for systems
of cascaded amps, mixers, filters, etc.  The results are displayed in both
graphical and tabular form, including system noise figure, overall gain,
maximum input level etc.  Different gains can be specified for signal and
noise.  'User(s)' is an optional user signal level in dBm.  Up to 2 user
input levels can be specified for inclusion in the plot.  'Name' is the
name (entered as a string) of a 7 row by N column data matrix containing
the data for N stages:

    Name[1;] = Noise Figure (dB)
    Name[2;] = Signal Gain (dB)
    Name[3;] = Output 2 Tone Third Order IM Intercept (dBm) *
    Name[4;] = Output Signal 1 dB Gain Compression Point (dBm) *
    Name[5;] = Noise Bandwidth (MHz)
    Name[6;] = Noise Gain (dB)
    Name[7;] = Output Noise 1 dB Gain Compression Point (dBm) *
```

77

```
The function also requires a global variable 'NameTYPE' containing
the labels for each stage.  The required data variables can be
created or added to using the 'MAKEDATA' function, or the stage
parameters can be edited using the 'EDITDATA' function.  Once a
screen plot is complete, typing <SHIFT>P exits to the text screen
and prints the plot.  Pressing any other key exits without printing.
```

```
        EXPLAIN 'DPRINT'
<User(s)> DPRINT 'Name' -- Prints Dynamic Range Analysis W/O Screen Display
This function is identical to DPLOT, only it sends the output directly to a
printer, without creating a screen display.  This allows users without
supported video systems to create plots.  For input and usage details,
run: EXPLAIN 'DPLOT'
```

14.2 GLOSSARY

Although most of the input and output terms are fairly common, some may be unfamiliar. A brief description of the less common terms and how they are calculated is given below:

Input Terms:

Noise Gain: This is listed separately from the Signal Gain to allow for the effects of signal processing. For example, averaging can be described as a stage with a Signal Gain of 0 dB, and a Noise Gain of -$10 \times Log(N_{avg})$.

Output Noise 1 dB Gain Compression Point: Because noise has spikes which are much higher than the average power, the 1 dB gain compression point will be significantly lower for noise than for a CW signal. In the absence of more detailed information, a noise compression point ~10 dB lower than the signal compression point is a good rule of thumb for WGN.

Output Terms:

Spot Noise Figure: This is the standard cascaded-stage noise figure for the complete system, based on the noise figure and noise gain of all of the stages.

Average Noise Figure: This is similar to the Spot Noise Figure, but it is normalized over the noise bandwidth of the system. In a typical receiver design where the narrowest bandwidth is at the final stage, the two noise figures are the same. However, if the stage that sets the noise bandwidth is not the last stage, the average noise figure will increase because of the wider bandwidth noise contributed by subsequent stages.

Maximum Wideband Input: This is equivalent to the Input Signal for 1 dB Gain Compression listed in the output Summary, only it is calculated using noise gains and noise output compression points.

Stage Input Equivalent Noise Level: This is the total noise power (in the system noise bandwidth) referred to the input of each stage based solely on the noise figure of that stage. It is NOT based on a running calculation of noise figure of a given stage combined with subsequent stages.

14.3 DEMO PLOT

There are variables in the workspace to produce a demonstration plot. Figure 14.1 shows the result of running: $^{-}35$ $DPLOT$ $'DEMO'$. The boxes above the data for each stage allow the user to draw in a symbol (amplifier, mixer, pad etc.) to provide a schematic representation of the system.

14.4 VDI PLOTTING

This workspace uses APL2000's VDI (Virtual Device Interface) graphics functions. This is only available in APL2000's $APL+PC$ or $APL+DOS$, not in $APLSE$. In order to get the high resolution required by the detail in these plots, special video drivers are needed. The only graphics devices that are currently supported are the ones based on the Tseng Labs ET4000 chip set. This includes the newer ET6000 chips, which are backward compatible. The $DPLOT$ function should NOT be used if you have another type of video card. Fortunately, the print driver will work with any HP LaserJet compatible printer. For a general description of VDI graphics and how to set up your computer to work with it, see the variable '$VDIHELP$'.

14.5 ACKNOWLEDGMENTS

The output format and analysis are largely based on a mainframe workspace written by David Hodsdon.

DYNAMIC RANGE PLOT FOR DEMO   18:58 11/17/96

LEVEL IN DBM

	PRE-AMP	AGC	CUPLER	PWR DIV	MIXER	ZFILT	IF AMP	SINC	CPL PAD	GAASAMP
NOISE FIG,DB	2.00	17.00	.50	4.00	8.50	1.00	9.00	2.00	4.00	4.50
SIG GAIN,DB	.00	20.00	-.50	-4.00	-8.50	-1.00	20.00	-2.00	-4.00	23.00
IP3 OUT,DBM		15.00			20.00		18.00			26.00
SIG COMP,DBM		5.00			-12.50		5.50			13.00
NOISE BW,MHZ	1600.00	1600.00	1600.00	1500.00	1500.00	500.00	3200.00	100.00	1500.00	3500.00
NOISE GAIN,DB	.00	20.00	-.50	-4.00	-8.50	-1.00	20.00	-2.00	-4.00	23.00
NOISE COMP,DBM		-3.00			-22.50		-4.50			3.00

SUMMARY:
SPOT NOISE FIGURE	=	17.22 DB
AVERAGE NOISE FIGURE	=	17.20 DB
INPUT SIGNAL FOR 1 DB GAIN COMPRESSION	=	-30.00 DBM
SIGNAL GAIN	=	43.00 DB
MAXIMUM WIDEBAND INPUT	=	-29.22 DBM
NOISE GAIN	=	43.00 DB
NOISE BANDWIDTH	=	100.00 MHZ

KEY
——— MAXIMUM INPUT SIGNAL
·········· EQUIVALENT SYSTEM NOISE LEVEL, 100.0 MHZ BW
-··-··- MAXIMUM WIDEBAND SIGNAL
-------- -35 DBM USER
+ STAGE OUTPUT 1 DB GAIN COMPRESSION
* STAGE INPUT EQUIVALENT NOISE LEVEL, 100.0 MHZ BW

Figure 14.1. Dynamic range DEMO plot

80

15. *RUFILSYN* AND *FRMRUFLT* WORKSPACES

LC FILTER SYNTHESIS & *MARTHA* CONVERTER

15.1 INTRODUCTION

'*RUFILSYN*' is a PC *APL* compatible version of the Rutgers University *FILTER* workspace [15.1]. Like *LLFILTER*, it also does lumped-element filter synthesis, but it can synthesize elliptical filters and filters with arbitrary responses that the *LLFILTER* workspace cannot. The functions in the workspace have not been documented for use with the '*EXPLAIN*' system, but there is an 85 page manual. A limited number of copies of this manual are available, and more could be photocopied if demand warrants it. *RUFILSYN* does not produce network descriptions directly compatible with *MARTHA*, but the '*FRMRUFLT*' workspace contains a conversion function to fill this void.

15.2 ELLIPTICAL FILTER DESIGN EXAMPLE

The process for designing filters in *RUFILSYN* is a bit more complex than using the Lincoln Laboratory functions. Its real strength is in designing complex filters with a fair amount of user interaction, and little attention was paid to streamlining the user interface. Designing a filter requires first calculating the poles and zeroes of the required transfer function. Using the results of this process, you can then synthesize the component values. One nice feature is that *RUFILSYN* creates schematics of the resulting filter through clever use of the *APL* character set. It was written as a stand-alone package, and a plotting system is included for checking the resulting frequency response. Because of its mainframe heritage, the plotting system uses *APL* text characters and looks a bit crude by today's standards. Below is an example session showing the design and analysis of a 3rd order 50 MHz bandwidth elliptical low-pass filter.

```
      S22E←ELLIP

ENTER AMAX, AMIN, FP(END OF PASSBAND), FS(START OF STOPBAND)
□:
      .1 20 50E6 100E6
ENTER 1 FOR TRANSFORMED ELLIPTIC BANDPASS FILTER, 0 OTHERWISE
□:
      0

ORDER OF THE ELLIPTIC FILTER, N= 3
REFLECTION ZEROS IN COMPLEX FORM (STORED IN RZERO):
      0 0 0 4.4052E7

TRANSMISSION ZEROS IN COMPLEX FOFM (STORED IN TZERO):
      0 1.135E8

ZETAF= ‾0.12464 (9.0433 DB)
ENTER 1 TO REALIZE THE FILTER, 0 TO EXIT FROM ELLIP
□:
      1

★★★AN ITERATIVE SOLUTION TO THE FELDTKELLER EQUATION FOLLOWS
THE NATURAL FREQUENCIES (OR REFLECTION ZEROS) IN COMPLEX FORM ARE:
```

‾5.5838E7 0 ‾1.9093E7 6.0895E7

USE FUNCTIONS "EL←PORT1 SI .... SI PORT2 S22" TO REALIZE THE FILTER
YOU NEED 1 FUNCTION S3   ; 0 FUNCTION S4
          1 FUNCTION S5 S3; 0 FUNCTION S5 S4

      ELE←PORT1 S3 S5 S3 PORT2 S22E

ENTER 0 IF THE LAST BRANCH SHOULD BE SHUNT, 1 IF IT IS IN SERIES
□:
      0
ENTER LOAD CONDUCTANCE
□:
      .02
ENTER FINITE NONZERO TRANSMISSION ZERO ASSOCIATE WITH EACH S5,
AND ENTER 0 ASSOCIATE WITH EACH S0, S3 OR S4
□:
      0 1.135E8 0

ELEMENT VALUES ARE:

```
|           |
|---∧∧∧---|        2.00E‾002                          G
|           |
|           |
|----||----|        5.70E‾011                          C
|           |
|      -----
|     |   ⊃
|     ---  ⊃        1.32E‾011      1.49E‾007      C,L       TRAP FREQ.
|     ---  ⊃                                              1.135E0008
|     |   ⊃
|      -----
|           |
|----||----|        5.70E‾011                          C
|           |
|           |
|---∧∧∧---|        2.00E‾002                          G
```

USE FUNCTION "RESP EL" TO ANALYSE THE FILTER IN VARIABLE EL
TO REALIZED THE FILTER WITH DIFFERENT CONFIGURATIONS USE FUNCTIONS
    "EL←PORT1 SI ..... SI PORT2 S22"

      RESP ELE

ANALYSIS

ENTER FREQUENCY RANGE AND SPACING (E.G. 85,100,1) FOR PLOT
□:
      0 300E6 10E6

82

```
FIGURE: THE MAGNITUDE OF S21 IN DB
                        \
                          ⁻40.0      ⁻30.0      ⁻20.0      ⁻10.0        .0
F          .00    .000E0000|          |          |          |          ★
R   10000000.00  ⁻2.880E⁻002|                                          ★
E   20000000.00  ⁻8.409E⁻002|                                          ★
Q   30000000.00  ⁻9.357E⁻002|                                          ★
U   40000000.00  ⁻2.027E⁻002|                                          ★
E   50000000.00  ⁻1.000E⁻001|                                         ★
N   60000000.00  ⁻1.365E0000|                                        ★
C   70000000.00  ⁻4.998E0000|                                   ★
Y   80000000.00  ⁻1.030E0001|                             ★
    90000000.00  ⁻1.645E0001|                        ★
   100000000.00  ⁻2.401E0001|                  ★
   110000000.00  ⁻3.815E0001| ★
   120000000.00  ⁻3.494E0001|     ★
   130000000.00  ⁻2.877E0001|          ★
   140000000.00  ⁻2.639E0001|             ★
   150000000.00  ⁻2.520E0001|              ★
   160000000.00  ⁻2.455E0001|              ★
   170000000.00  ⁻2.421E0001|               ★
   180000000.00  ⁻2.405E0001|               ★
   190000000.00  ⁻2.401E0001|               ★
   200000000.00  ⁻2.405E0001|               ★
   210000000.00  ⁻2.414E0001|               ★
   220000000.00  ⁻2.426E0001|               ★
   230000000.00  ⁻2.441E0001|               ★
   240000000.00  ⁻2.458E0001|              ★
   250000000.00  ⁻2.477E0001|              ★
   260000000.00  ⁻2.496E0001|              ★
   270000000.00  ⁻2.516E0001|              ★
   280000000.00  ⁻2.536E0001|              ★
   290000000.00  ⁻2.557E0001|             ★
   300000000.00  ⁻2.577E0001|          |  ★       |          |          |
```

ENTER 1 FOR THE MAGNITUDE OF S21
       2 FOR PHASE AND GROUP DELAY; 0 FOR THE END OF ANALYSIS
▢:
       0

15.3 CONVERSION TO *MARTHA* FORMAT

The workspace *FRMRUFLT* contains the function '*FROMFILTER*' which allows converting the filter descriptions from *RUFILSYN* into a *MARTHA* compatible form. The fundamental way that the two programs model filters is a bit different, so a certain amount of user input is required. Once a filter is designed using *RUFILSYN*, the results should be saved. *FROMFILTER* requires a number of *MARTHA* functions, so both *FROMFILTER* and the filter network should be copied into a workspace containing *MARTHA* before attempting the conversion process. *FROMFILTER* was written before the advent of the *EXPLAIN* system, but there is a help variable '*HOWFROMFILTER*' included in the workspace:

VARIABLE 'HOWFROMFILTER' IN RUFILSYN WORKSPACE

This function converts filters designed using the Rutgers University 'FILTER' package into MARTHA form. The 'FILTER' package is in the RUFILSYN workspace, and details of its usage are contained in the manual: FILTER - An APL Filter Design Package, by F.C. Liu and T.G. Marshall, Jr., 1977. The 'FROMFILTER' function must be executed in a workspace containing MARTHA. The syntax is:

        Net←Left FROMFILTER Fltr

Where 'Fltr' is the filter as generated by the FILTER package, and 'Left' contains auxiliary information necessary in the conversion. The result, 'Net', is either the MARTHA network itself, or, in cases to be described, the name of a function which, upon execution, produces this network.

'Fltr' is a 4-column numerical matrix, with one row for each filter section. Fltr[;1] is either 0, 3, 4, or 5, depending on the type of section. Fltr[;2 3 4] are either ignored (for some sections) or are element values. For a detailed description of this format, see pages 11 and 12 of the FILTER program manual.

The network specified by this matrix is incomplete, in the sense that both 'Fltr' and its dual are represented the same way in FILTER, whereas in MARTHA they are not. During the conversion, a choice between the two possible filters must be made. In addition, in FILTER, the source and load resistances are included in the matrix, whereas in MARTHA they are not included in the network definitions. At times it may be useful to include the source and/or load impedances in the MARTHA network definition, or sometimes a network coming from the FILTER package may not have a source or load specified. Therefore, an option is necessary to allow (but not force) the setting of ZG and ZL in MARTHA. Finally, the desired result may be a MARTHA network, or a function that can be executed to produce the network, which can be displayed and/or modified. Thus there are three conversion options that must be specified. These are: (1) which of the two dual networks to produce; (2) whether source and load impedances, if present, are to be set or included in the network definition (or neither); and (3) whether to produce a network or an APL function.

The variable 'Left' sets these options. It is a character vector beginning with one of the four strings:

        'LSE' OR 'LSH' OR 'RSE' OR 'RSH'

which stands for 'Left (or Right) Series (or Shunt)'. If 'LSE' is specified, then the left most section will be treated as a series section. Although four choices are possible, only two possible network configurations exist. Thus if the number of sections is even, the 'LSE' and 'RSH' are equivalent, whereas if there is an odd number of sections, then 'LSE' and 'RSE' are equivalent. After the three letter code, the following characters may optionally appear:
< > × [ ] ▯ or ∇. If '∇' appears, it must be followed by a valid APL

function name. The result of the conversion will be the implicit creation of a function by that name, and the explicit return of that name. Otherwise, a network is created and returned directly.

If the deduced values for ZG and ZL are not to be set, and their values are not to be part of the explicitly returned network, the mode symbol '<' (for ZG) or '>' (for ZL) should be used. An abbreviation for both '<' and '>' is 'x'.

If the values for ZG and/or ZL are to be incorporated into the explicitly returned network, and the variables ZG and ZL are not to be set, the mode symbols '[' (for ZG) or ']' (for ZL) should be used. An abbreviation for both '[' and ']' is '□'.

If the values for ZG and ZL are to be set and not appear in the network, no mode symbol is used. This is the default condition. Note that if '[' appears then '<' is allowed but adds nothing; similarly if ']' appears then '>' may also, but is not necessary.

If the mode symbol '∇' appears and any of the mode symbols: < > x [ ] OR □, is used to inhibit the setting of ZG and/or ZL, then the function created will include a comment line setting the values of ZG and/or ZL.

In all of these cases, the deduced value of ZG is the resistance of the lossy part of the first section, if it is non-zero and the type of the first section is 3 or 4. If the first section is of type 0 or 5, or if the resistance or conductance is zero, then no setting of ZG occurs. Thus a resistor of value 0.0000000001 Ohm sets ZG, but a resistor of value 0 Ohm does not. A similar criterion is used on the last section of the filter for ZL.

15.4 CONVERSION FROM MAINFRAME TO PC *APL*

The *RUFILSYN* workspace was originally written for use on mainframe computers, and required extensive modifications to get it working with the PC *APL*. For the morbidly curious and anyone attempting future conversions, the gory details are detailed in the workspace. The latent expression, □LX, runs the '*DESCRIBE*' function, which displays the following:

WELCOME TO APL FILTER PACKAGE. FOR THE DETAILS, SEE MANUAL:
FILTER - An APL Filter Design Package, F.C. Liu and T.G. Marshall, Jr.
Rutgers University and David Sarnoff Research Center, 6/24/77

Modified By D.W. White to run under APL2000 APL+PC
M.I.T. Lincoln Laboratory, 7/31/91

The changes made to allow operation under APL+PC should not generally affect the use of the functions, but they are described below in case of difficulties:

1) Replaced function calls: 'ibe 26' with '1↑□LC'. 'ibe' was a function used to replace the old 'I' (I-beam) system function.

2) *Eliminated use of* 101 *☐SVO stack processor in these functions:*

> PROTOTYPE - *Now calls TRANSFORM and TOPBAND using optional*
> *left arguments instead of using stack*

> TRANSFORM - *Modified to accept an optional left argument of:*
> *1st Element (FE), Transform Type, Zo, Fc*
> *Also now calls BANDTRAN with optional left arguments*
> *instead of stack.*

> BANDTRAN - *Added optional elements to left argument. Left*
> *arg. is now: 1st Element (FE), [Zo, Fc] (optional)*

> TOPBAND   - *Now accepts an optional left argument of: Zo, Fc*

3) *Modified READ function to eliminate references to CMS disk files,*
and removed several lines of 'commented-out' code.

4) *Modified ELMATRIX function to correct differences between PC*
and Mainframe use of prompts with ☐. The Mainframe places
spaces in the result, and APL+PC echoes the prompt string.
The prompt string can be eliminated by using ☐ARBIN 10

5) *Adjusted spacing in DISPLAY and GRAPH to account for extra digit*
in APL+PC's ⅋ exponents. Also added branch labels to GRAPH.

The modified code makes use of ◊, ☐LC, and ☐NC, all of which should
be supported in most standard APL's. The problem with ☐ appears to
be much more system dependent, and will have to be dealt with on a
case-by-case basis. ELMATRIX is the only function where this
construct appears to be used. Minor adjustments in output displays
may be required because of variations in ⅋ exponent length.

15.5 ACKNOWLEDGMENTS

The *RUFILSYN* package (under the name *FILTER*) is, of course, the work of F.C. Liu, and T.G. Marshal, Jr.. The author of the *FROMFILTER* conversion function has unfortunately been lost to the mists of time.

15.6 REFERENCES

15.1   F.C. Liu, and T.G. Marshal, Jr. "*FILTER* - An *APL* Filter Design Package," Rutgers University E.E. Dept., 1977

16. INDEX

KEY: fn = function, ws = workspace, var = variable

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE 1 December 2003	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE

LLAMA (Lincoln LaboratoryAdvanced MARTHA Applications) Software Manual

5. FUNDING NUMBERS

C—F19628-00-C-0002

6. AUTHOR(S)

D.W. White

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lincoln Laboratory, MIT
244 Wood Street
Lexington, MA 02420-9108

8. PERFORMING ORGANIZATION REPORT NUMBER

TR-1088

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force ESC/XPK
5 Eglin Street
Hanscom AFB, MA 01731

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ESC-TR-2003-056

11. SUPPLEMENTARY NOTES

None

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

For the past 25 years or more, a number of staff members at MIT Lincoln Laboratory have made extensive use of the APL computer language to solve a variety of problems, primarily in the area of radio frequency and microwave circuit design. This was aided and inspired by the availability of the *MARTHA* software package, which is a collection of *APL*-based circuit analysis functions developed by Professor Paul Penfield Jr. at MIT.

The Lincoln Laboratory Advanced *MARTHA* Applications (or *LLAMA* for short) is a set of 15 workspaces (a collection of *APL* functions) developed primarily in conjunction with *MARTHA*. Many of the workspaces are an extension of *MARTHA*, and allow the use of a new circuit elements or new types of analysis. A number of workspaces are devoted to filter synthesis, using both lumped elements and coupled microstrip transmission lines. Other workspaces are aimed toward RF system design, including mixer-spur and dynamic-range analysis.

This manual is intended to provide more formal documentation for this resource than has previously been available. It is shoped that it will allow new users to quickly make use of all that *APL* and *MARTHA* have to offer, as well as providing a concise, well-indexed reference for the more experienced user.

14. SUBJECT TERMS

15. NUMBER OF PAGES
107

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Same as Report